# The Resource Description Framework and its Schema

Fabien Gandon, Reto Krummenacher, Sung-Kook Han, Ioan Toma

# The Resource Description Framework and its Schema

Fabien L. Gandon, INRIA Sophia Antipolis
Reto Krummenacher, STI Innsbruck
Sung-Kook Han, STI Innsbruck
Ioan Toma, STI Innsbruck

## 1. Abstract

RDF is a framework to publish statements on the web about anything. It allows anyone to describe resources, in particular Web resources, such as the author, creation date, subject, and copyright of an image. Any information portal or data-based web site can be interested in using the graph model of RDF to open its silos of data about persons, documents, events, products, services, places etc. RDF reuses the web approach to identify resources (URI) and to allow one to explicitly represent any relationship between two resources. Such statements can come from any source on the web and be merged with other statements supporting world-wide data integration. Using and reusing URIs, anyone can say anything about any topic, anyone can add to it, and so on. Additionally, using RDFS, one can define domain-specific classes and properties to describe these resources and organize them in hierarchies. These schemas are also published and exchanged in RDF. RDF not only provides a graph model to publish and link data on the web, it also provides the foundational shared data model on which other capabilities are built: querying (SPARQL is built on top of RDF), embedding (RDFa and GRDDL rely on the RDF model), and reasoning (RDFS and OWL are defined on top of RDF). Semantic web is a web to link data and share the semantics of their schemas. RDF provides a recommendation to publish and link data. RDFS provides a recommendation to share the semantics of their schemas. The couple RDF & RDFS is also reused in several other activities of the W3C.

## *2.* Scientific and Technical Overview

### *2.1. A Model for a Resource Description Framework*

#### *2.1.1. Resource descriptions*

The Semantic Web as it is emerging and evolving today has become an ever growing cloud of interlinked data. As a matter of fact, the Semantic Web materializes as a Web of Linked Data. Annotated objects are connected to other annotated artifacts that are identifiable on the Web; spanning a network of connected resources whose links carry a lot of hidden knowledge. Discovering such implicit knowledge and making it explicitly available to interested parties via query engines and Web applications is the main driver of current Semantic Web research. Besides a cloud of facts that are available through the Semantic Web, more and more ontologies are published that allow making sense out of the Web of Data. While ontologies – in the simplest case written as RDF Schemas (RDFS) – are thus very important for interpreting data and for deducting knowledge out of available facts, RDF is the *de facto* standard for metadata and annotating things on the Web, and hence the lingua franca of open data. But why RDF, and what for?

The abbreviation RDF stands for the "Resource Description Framework" in the sense that:

*Resources* are a core concept on the semantic web: everything one might refer to is considered a resource; e.g., a Web page, an image, a videos, but also a person, a place, a device, an event, an organization, a product, or a service. More technically speaking, everything that can be identified by a URI can be considered a resource.

*Descriptions* of resources are essential for understanding and reasoning about them. In the most general case, a description is a set of attributes, features, and relations concerning the resource.

The *Framework* means it provides models, languages and syntaxes for these descriptions.

In short, RDF provides a standard data structure and model to encode data and metadata about any subject on the web.

### *2.1.2. Triples as atoms of knowledge*

RDF graphs are constructed out of so-called RDF triples (Figure 2). RDF triples describe and connect objects via the combination of resources, properties, and property values; such triples are also referred to as statements. The resource is the subject of the statement; the property is the predicate of the statement; and the property value is the object of the statement. Therefore, the basic data structure of RDF is the triple of the form `<subject, predicate, object>`.

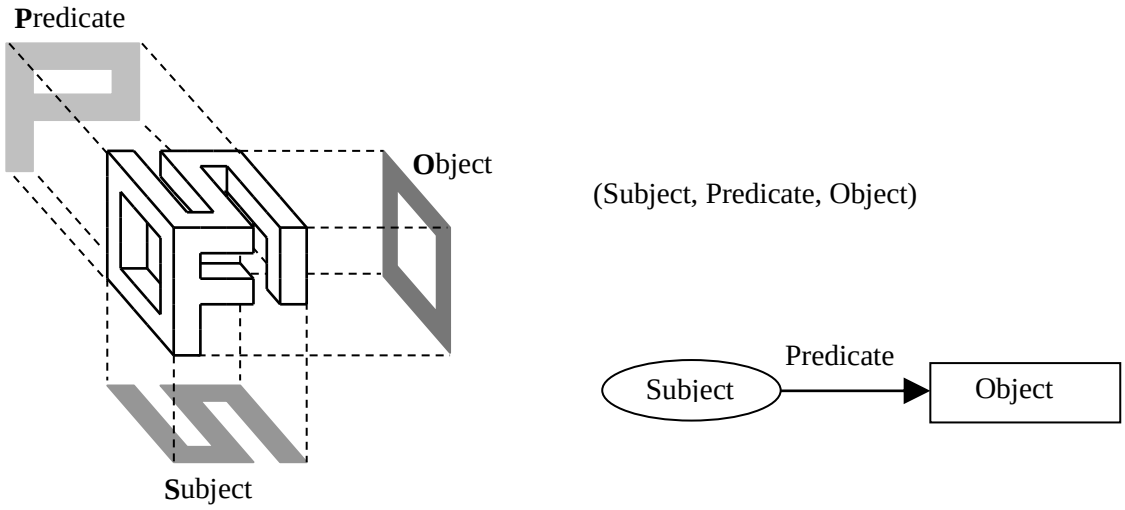



*Fig. 2. The RDF triple: the atom of knowledge on the semantic web inspired by "Gödel, Escher, Bach: An Eternal Golden Braid" (Hofstadter, 1999)*

For example, the assertion "Fabien has written a page doc.html about Music" can be broken down into two RDF statements about the document: (doc.html, author, Fabien) and (doc.html, theme, music). Here again, the resource doc.html is the subject, the attribute theme is the predicate and the third element (Music) is the object of the statement.

Being a web-oriented framework, RDF identifies resources and properties using URIs, optimally even URLs that allow dereferencing the identified web resources and to discover RDF descriptions – this process is pretty much comparable to the traversal of hyperlinks in HTML Web documents. As stated previously, shared resources across subjects and objects are one of the fundamental principles that allow for constructing RDF graphs. For this reason, URIs are found as the subjects and objects of triples, and in RDF also as unique identifiers for predicates. In RDF, besides being URIs, objects can also take the form of so-called literals *i.e.* arbitrary typed or untyped strings. Reconsidering the example triple (doc.html, author, Fabien), the subject would be an HTML resource on the web, while the property would be identified with a globally unique URI. The object of the given triple could, as explained above, be either a resource itself or a string literal to form one of the following triples: (<doc.html> <author> <Fabien>) or (<doc.html> <author> “Fabien”). In the former case, <Fabien> is a resource and hence a node in the RDF graph that could be dereferenced to discover further facts about the object Fabien; in the latter case, the object is given as a string and becomes a stub in the RDF graph, as the literal “Fabien” does not denote a shared resource.

RDF triples are axiomatic statements, facts that can be seen as binary predicates in logics. An important aspect of the logical view of the RDF triples is that RDF makes an open-world assumption: as opposed to the closed world assumption of classical systems the absence of a triple is not significant; *i.e.*, if a triple does not explicitly claim a fact, it does not mean that the fact could not be true. In other words, the RDF semantics assumes that whatever is not explicitly stated could be true, while in relational models the assumption is the other way around: facts that are not explicitly claimed are false. In the example below (Table 1), the fact that one only knows the authors Fabien and York does not mean that there are no other authors; it only means that there are at least the two named authors.

### *2.1.3. A graph-oriented data model*

RDF triples can also be seen as two vertices and one arc of a graph describing and linking resources. More technically speaking, RDF is a decentralized data representation model relying on distributed triples that form a global graph. The identity of the resources is based on the URI mechanism: if two resources have the same URI they are one and the same node in the graph. This extremely simple and powerful data model representation mechanism has recently found adopters in a large range of domains. The World Wide Web Consortium W3C, for instance, uses RDF in many other activities than the semantic web only; e.g., the privacy preference language of P3P uses RDF, or various multimedia standards benefit from RDF. Although RDF was initially published as a base layer of the semantic web standards, it can now be observed that its graph structure can more generally be leveraged as one of the two options (XML infosets and RDF graphs) for data structures in all the known web standards (Figure 1).

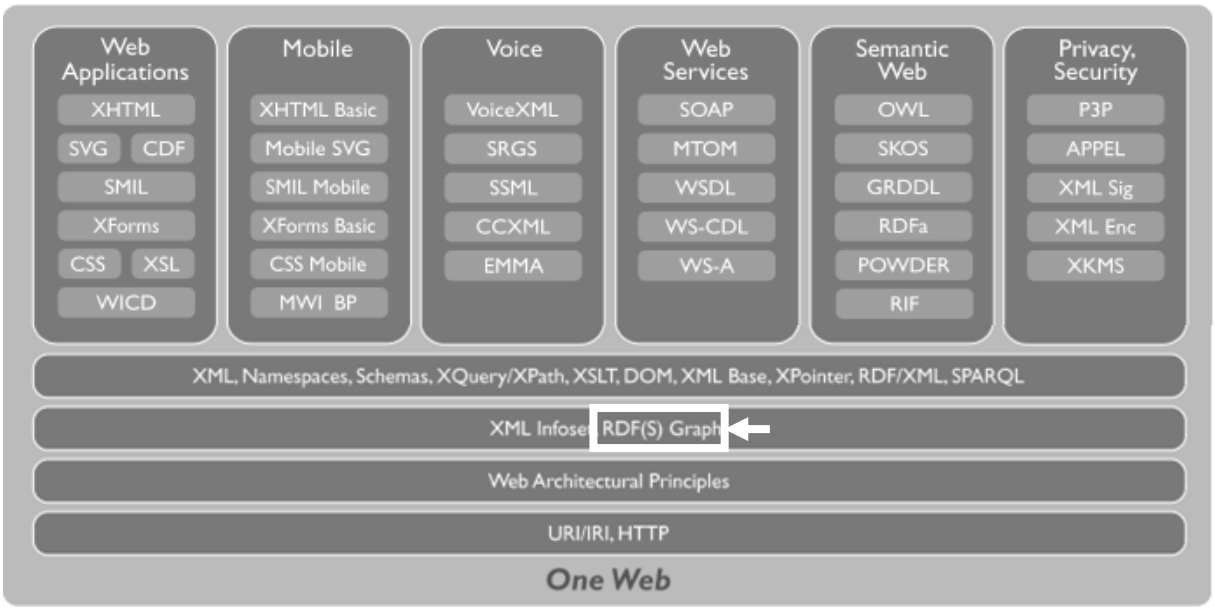



Fig. 1 RDF graphs at the foundations of the recommendation stack at W3C © [22]

An RDF graph is:

a multi-graph, a graph that can contain both multiple edges and loops between its vertices.

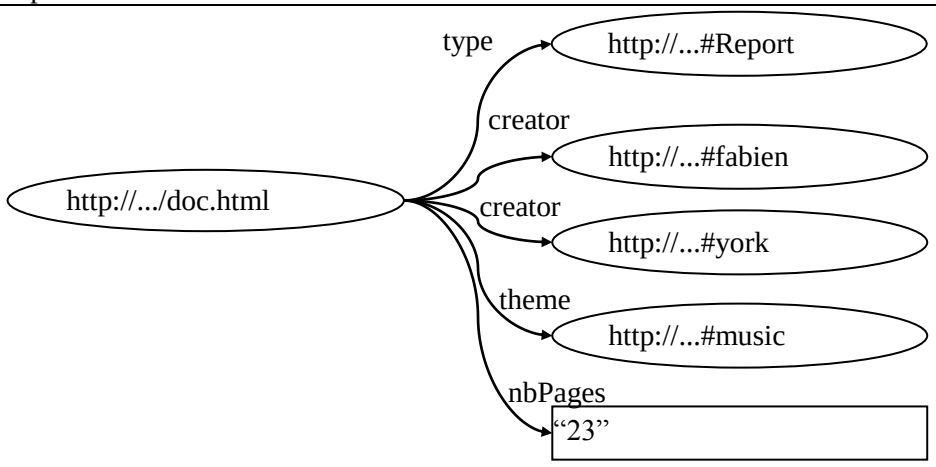
a directed graph, every edge is oriented going from the end-vertex representing the subject to the end-vertex representing the object.

a labeled graph, RDF assigns a unique label to the edges and vertices of a graph; edges are labeled with URIs, vertices are labeled with URIs, blank node identifiers or literals. An RDF blank node is an RDF node that itself does not contain any data, but serves as a parent node to a grouping of data.

From the graphical notation point of view, RDF graphs are directed labeled graphs where resource nodes are depicted as ovals, predicates label the directed arcs depicted as arrows and literals are stubs and depicted as rectangles.

One of the strongest assets of RDF is its graph structure. It allows RDF to offer a very flexible and highly extensible data model: anyone can add an arc anywhere in the graph whenever there is a new statement to make about an already present resource. Reusing an existing URI as the identifier of the subject or the object of the new triple allows for sharing resources and enlarging the RDF graph.

*Table 1. Simple RDF example.*

| "The report doc.html has for authors Fabien and York, has for theme Music and is 23-page long." |
|---|
| English |
| Report(doc.html)<br>creator(doc.html,Fabien)<br>creator(doc.html,York)<br>theme(doc.html,Music)<br>nbPages(doc.html, 23) |
| Logic predicates |
| (doc.html , type , Report)<br>(doc.html , creator , Fabien)<br>(doc.html , creator , York)<br>(doc.html , theme , Music)<br>(doc.html , nbPages , "23") |
| Triples |
|  |
| Graph |

### *2.1.4. Identifying conceptual vocabularies through namespaces*

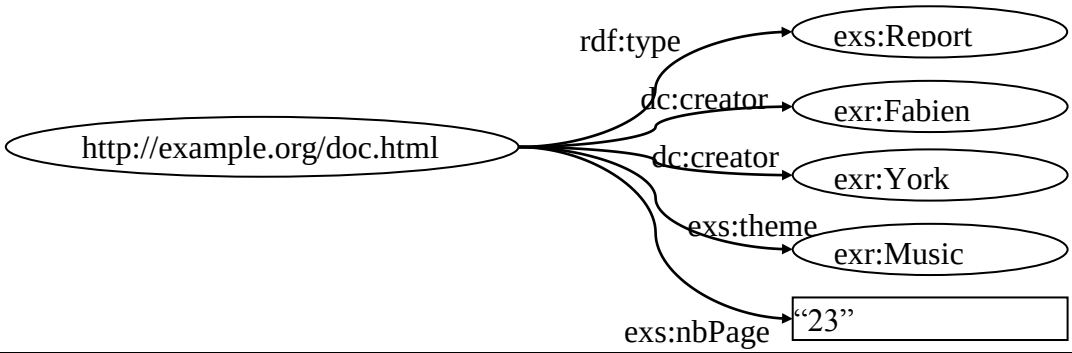
As stated previously, URIs are used to identify a diversity of resources. One special case is when a URI is used to identify a set of terms, a vocabulary, a schema; in that case the URI is called a namespace. Namespaces are used in particular to identify the schemas declaring the types of resources and types of relations used to label RDF graphs. In XML documents namespaces are associated to prefixes in order to shorten the resource identifiers in the document, locally using the prefix instead of the full URI. For instance, RDF provides an elementary typing primitive (`rdf:type`). The `type` predicate thus belongs to the core RDF vocabulary which is identified by the URI http://www.w3.org/1999/02/22-rdf-syntax-ns# which is often associated with the prefix `rdf`. Consequently, the `type` predicate can be identified as `rdf:type` instead of http://www.w3.org/1999/02/22-rdf-syntax-ns#type.

In this chapter the following namespaces and prefixes will be used:

| Prefix of the namespace | URI of the namespace |
|---|---|
| rdf | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs | http://www.w3.org/2000/01/rdf-schema# |
| foaf | http://xmlns.com/foaf/0.1/ (a schema to describe peoples) |
| dc | http://purl.org/dc/elements/1.1/ (schema for documents) |
| xsd | http://www.w3.org/2001/XMLSchema# (datatypes for literals) |
| exs | http://example.org/schema# (fictional schema namespace) |
| exr | http://example.org# (fictional namespace of resources) |

Using these namespaces and their prefixes one can now rewrite properly the example of Table 1:

*Table 2. Rewritten example using namespaces and URIs.*

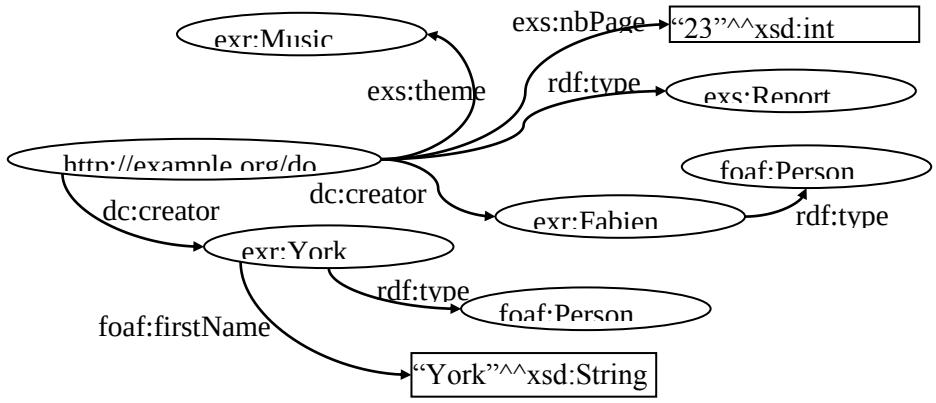
| |
|---|
| (http://example.org/doc.html , rdf:type , exs:Report)<br>(http://example.org/doc.html , dc:creator , exr:Fabien)<br>(http://example.org/doc.html , dc:creator , exr:York)<br>(http://example.org/doc.html , exs:theme , exr:Music)<br>(http://example.org/doc.html , exs:nbPages , "23") |
| RDF Triples |
|  |
| RDF Graph |

### *2.1.5. Ontology-oriented typing and multi-instantiation*

The example shown in

Table 2 depicts how the RDF typing primitive `rdf:type` allows for attaching schema information in the form of one or more types to a resource: http://example.org/doc.html is declared as being a report *i.e.*, a resource of type `exs:Report`. One important and very powerful difference between typing in RDF and typing in most common object-oriented programming languages is that an RDF resource can belong to several unrelated classes; i.e.; RDF allows multiple types for a resource, a kind of multi-instantiation. Note that literals, although not denoting objects, can also be typed; e.g., primitive data values are mostly type using the XML Schema data types.

*Table 3. Typing all RDF resources and the literals.*

| |
|---|
| (http://example.org/doc.html , rdf:type , exs:Report)<br>(http://example.org/doc.html , dc:creator , exr:Fabien)<br>(http://example.org/doc.html , dc:creator , exr:York)<br>(exr:York, foaf:firstName, “York”^^xsd:String)<br>(http://example.org/doc.html , exs:theme , exr:Music)<br>(http://example.org/doc.html , exs:nbPages , “23”^^xsd:int) |
| RDF Triples |
|  |
| RDF Graph |

### *2.1.6. Existential quantification of resources: blank nodes*

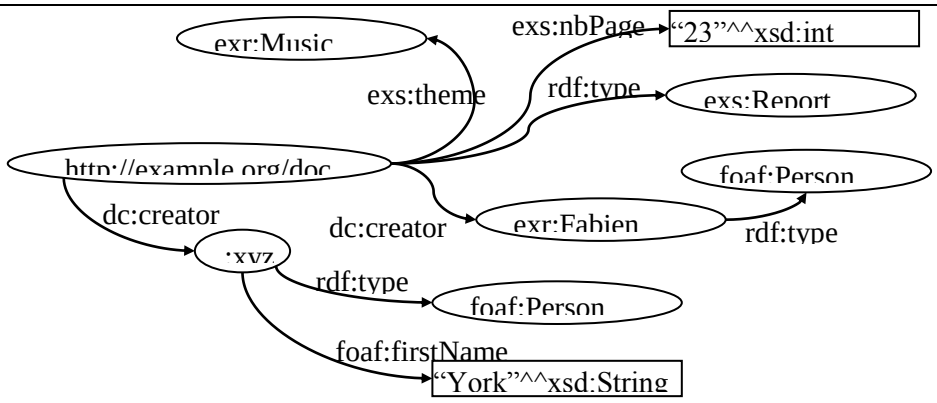
In all the examples so far, resources that could be associated with a URI in order to identify them on a global scale have been used. In some cases this unified identification of resources is not possible. There might potentially be situations, where it is known that there is some object but it is not known which particular instance one deals with. For such situations, RDF introduces the concept of blank nodes, also called bnodes in short. Having a blank node in an RDF graph means that the vertex representing the resource is unknown or anonymous; *i.e.*, the resource at hand is not identified by a URI. In

Table 4, the blank node is noted using an underscore instead of the namespace prefix (_:xyz). As such anonymous identifiers cannot assume a globally unique name, they can only be used inside one single RDF file; the one in which it was declared. This has the consequence that blank node resources in different graphs or files, although potentially carrying the same bnode ID (e.g., _:xyz) denote two distinct resources according to the RDF semantics. In summary, a blank node is like an existential quantification in logics, it means “there exists a resource such that…”. Taking a closer look at the example in

Table 4, and assuming that there is no URI provided for the author York, a blank node can be introduced that carries the semantics: there exists a resource such that it is a foaf:Person that has the first name “York”.

*Table 4. Introducing RDF blank nodes*

| |
|---|
| (http://example.org/doc.html , rdf:type , exs:Report)<br>(http://example.org/doc.html , dc:creator , exr:Fabien)<br>(http://example.org/doc.html , dc:creator , _:xyz)<br>(_:xyz, foaf:firstName, “York”^^xsd:String)<br>(http://example.org/doc.html , exs:theme , exr:Music)<br>(http://example.org/doc.html , exs:nbPages , “23”^^xsd:int) |
| RDF Triples |



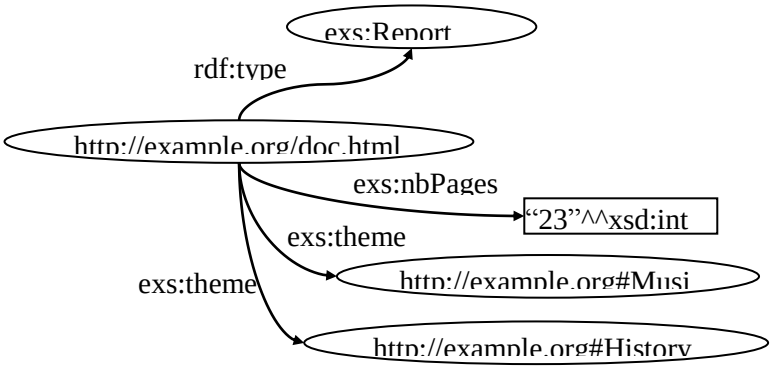

RDF Graph

Although blank nodes are allowed in RDF, they are discouraged in practice since they "break the graph"; they cannot be reused outside the RDF file they were declared and thus prevent anyone to simply extend and relate to the anonymously declared resource. In the example above everyone can add information about Fabien just reusing the URI expanded from `exr:Fabien`, however no one could add any information about York since there is no way to reference it outside the RDF file where the resource was declared. This clearly counteracts the idea of extensibility and reusability which are at the basis of the very successful Linked Open Data initiative (www.linkeddata.org). The problem of choosing the right naming scheme can be very difficult and holds many consequences; this is why good practices to design URIs are proposed [23].

The remainder of this chapter looks in more details at the building blocks of RDF, its syntax and semantics.

## *2.2. Serializing RDF Graphs*

As discussed earlier, RDF statements can be represented in directed graph structure, called RDF graph, which is very useful for representing RDF information and good for human understanding. The following shows a simple example of RDF graph which represents an object whose type is a report (`exs:Report`) and having properties such as the number of pages (`exs:nbPages`) and its theme (`exs:theme`).




RDF graph

However, RDF graph is an abstract model. There are several serialization formats for converting the abstract RDF graph into a concrete byte stream. Some of the most popular ones are RDF/XML, N-Triples, Turtle and N3.

### *2.2.1. RDF Graphs as XML trees*

RDF documents can be written in XML format and this format is called RDF/XML. RDF/XML is designed to be read and understood by computers and developers, but it is not designed for being read by end-users. RDF descriptions in RDF/XML are not supposed to be directly displayed on the web for other purposes than developing or debugging. By using XML, RDF documents can be exchanged between very different types of systems and applications.

The following figure shows two versions of RDF/XML serialization of the above RDF graph. An RDF/XML document has an root element `<rdf:RDF>` that declares this XML document is really an RDF document. This element usually also contains the declaration of the RDF namespace and other namespaces depending on the content. A simple description uses the `<rdf:Description>` element with the attribute `rdf:ID` (with a simple ID) or `rdf:about` (with a URI) to identify the resource being described and containing elements (properties) that describe the resource. To obtain a full URI the `rdf:ID` is expanded against the base of the XML document as a fragment i.e. a hash separates the URI of the base and the ID of the description. The `<rdf:Description>` and `<rdf:type>` elements can be merged as a class element for simplicity (e.g., `<exs:Report>` in the right of the figure). Property elements may also be declared as attributes of the description. Look at the difference of `exs:nbPages` property between the two versions.

For more information about RDF/XML syntax, please refer to the authoritative specification site [6, 17].

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:exs="http://example.org/schema#">
 <rdf:Description rdf:about="http://example.org/doc.html">
  <rdf:type rdf:resource="http://example.org/schema#Report"/>
  <exs:theme rdf:resource="http://example.org#Music"/>
  <exs:theme rdf:resource="http://example.org#History"/>
  <exs:nbPages rdf:datatype="http://www.w3.org/2001/XMLSchema#int">23</exs:nbPages>
 </rdf:Description>
</rdf:RDF>
```

RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:exs="http://example.org/schema#">
 <exs:Report rdf:about="http://example.org/doc.html" exs:nbPages="23">
  <exs:theme rdf:resource="http://example.org#Music"/>
  <exs:theme rdf:resource="http://example.org#History"/>
 </exs:Report>
</rdf:RDF>
```

RDF/XML (simplified)

Other non standard syntaxes like N-Triples, Turtle and N3.

### *2.2.2. Triple serialization in N-Triple*

N-Triples are a line-based, plain text format for serializing an RDF graph and follows US-ASCII encoding. It is designed to be a fixed subset of N3 and also subset of Turtle. That is, N-Triples is a simplified format of Turtle, which is also simplified from N3. N-Triples does not support prefix abbreviation and shorthand notations for statements, requires just one statement a line, and restricts a statement within one line. Each line except comment and blank lines consists of subject, predicate and

object separated by whitespaces and terminated by a dot (.). This simplicity is appropriate for applications with stream data but increases the size of the serialized output due to the extreme redundancy of namespace string. N-Triples is not efficient in terms of compression ratio of information.

The following figure shows an example of N-Triples serialization corresponding to the above RDF graph. The namespace http://example.org/schema# repeatedly appears in several resources and the same subject resource is also repeated in each statement.

For more information about N-Triples syntax, please refer to the authoritative specification site [21].

```
<http://example.org/doc.html> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://example.org/schema#Report> .
<http://example.org/doc.html> <http://example.org/schema#theme>
<http://example.org#Music> .
<http://example.org/doc.html> <http://example.org/schema#theme>
<http://example.org#History> .
<http://example.org/doc.html> <http://example.org/schema#nbPages>
"23"^^<http://www.w3.org/2001/XMLSchema#int> .
```

N-Triples

### *2.2.3. Triple serialization in Turtle*

Turtle (Terse RDF Triple Language) [7] is a compact textual format for serializing an RDF graph and less constrained than N-Triples but more simplified than N3. Turtle does not have the restriction of single-line statement of N-Triples anymore.

A Turtle document consists of a sequence of directives, triple-generating statements or blank lines. The following figure shows an example of Turtle serialization of the above RDF graph. The directives include the declarations of base URI (`@base` directive) and prefixes of namespaces (`@prefix` directive). Through these declarations, URIs can be abbreviated effectively such as `rdf:type` and `exs:Report`.

Turtle supports two additional abbreviating notations. One is for repeated subject and the other is for repeated subject-predicate pair. To abbreviate multiple statements with the same subject, Turtle provides semicolon (;) notation which makes it possible to list predicate-object pairs for the same subject after a semicolon. In the figure, the same subject `<http://example.org/doc.html>` of three statements occurs just once through the abbreviation by semicolon. Multiple statements with the same subject-predicate pair but different objects can be similarly abbreviated using comma (,) notation, as in the case of `exs:theme` property of the figure.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix exs: <http://example.org/schema#> .
<http://example.org/doc.html> rdf:type exs:Report ;
  exs:theme <http://example.org#Music> , <http://example.org#History> ;
  exs:nbPages "23"^^xsd:int .
```

Turtle

As in RDF/XML serialization, Turtle provides a simplification about the type of an individual. That is, the article `a` can be used instead of `rdf:type`. So, the first statement in the figure can be written as follows: `<http://example.org/doc.html>` `a` `exs:Report` ;

The several abbreviation notations supported by Turtle make Turtle the simplest and most concise serialization format. It can effectively reduce the size of serialized byte stream. In addition, Turtle is the most human-friendly and readable syntax.

For more information about Turtle syntax, please refer to the authoritative specification site [7].

## *2.3. Predefined sub-graph structures*

RDF provides primitives to build containers and collections to list things.

### *2.3.1. Open containers*

Containers are open groups and contain resources or literals and possibly duplicates. There are three types of containers:

`rdf:Bag` for an unordered group of resources or literals that may contain duplicate values e.g. to give the content of a parcel.

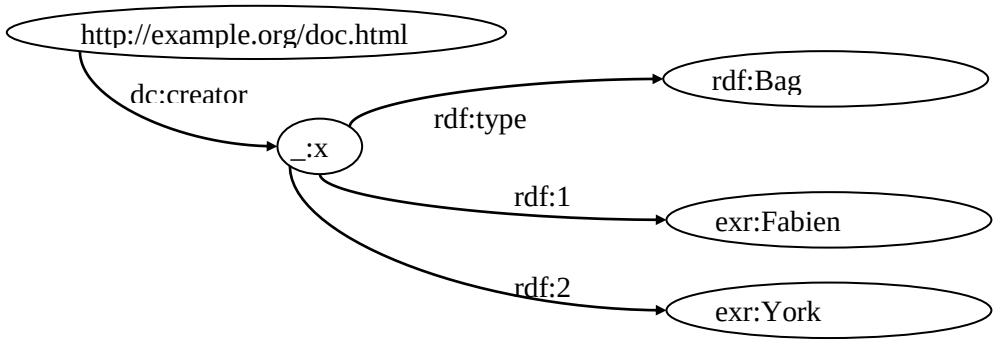
`rdf:Seq` for an ordered group of resources or literals that may contain duplicate values e.g. to list the winners of lottery in drawing order.

`rdf:Alt` represents a group of resources or literals that are alternatives i.e. only one of the values can be selected, used in particular for a single-valued property e.g. the title of a book in different languages.

In each case, a resource is typed as `rdf:Bag`, `rdf:Seq` or `rdf:Alt`, and, syntactically, the members of the container (the resources or literals it contains) are attached to this resource using the `rdf:li` property. This is a convenience property of RDF/XML to avoid having to explicitly number each membership relation but each occurrence of `rdf:li` is turned into a numbered property `rdf:_1`, `rdf:_2`, etc. to form the corresponding RDF graph. For instance, to give two authors of a document:

```
<exs:Report rdf:about="http://example.org/doc.html">
 <dc:creator>
  <rdf:Bag>
   <rdf:li rdf:resource="http://example.org#Fabien"/>
   <rdf:li rdf:resource="http://example.org#York"/>
  </rdf:Bag>
 </dc:creator>
</exs:Report>
```

RDF/XML



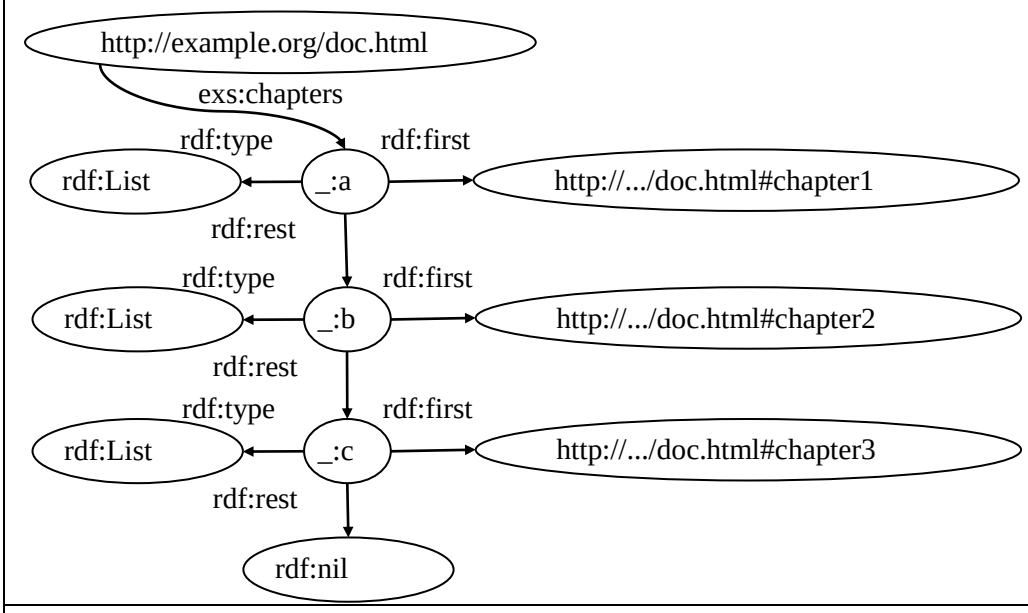

RDF Graph

### *2.3.2. Closed collections*

Containers remain open: there is no way to say that the only members a container contains are the one listed here. Collections, on the contrary, are closed lists of resources or literals, possibly with duplicates. RDF collections describe closed groups *i.e.* these groups contain only the specified members. The list of the members of a collection starts with a resource of type `rdf:List` with a property `rdf:first` pointing to the first member and a property `rdf:rest` recursively pointing to the list of the remaining members or to `rdf:nil` if one has reached the end of the list (as in LISP). The resource rdf:nil is an instance of rdf:List that can be used to represent an empty list or other list-like structure. Here again, RDF/XML provides a convenience notation: a collection is declared as nested elements directly given in a property that has the attribute `rdf:parseType="Collection"`. For instance to list the three chapters of a document in order:

```
<exs:Report rdf:about="http://example.org/doc.html">
 <exs:chapters rdf:parseType="Collection">
  <rdf:Description rdf:about="http://example.org/doc.html#chapter1"/>
  <rdf:Description rdf:about="http://example.org/doc.html#chapter2"/>
  <rdf:Description rdf:about="http://example.org/doc.html#chapter3"/>
</exs:chapters>
</exs:Report>
```

**RDF/XML**



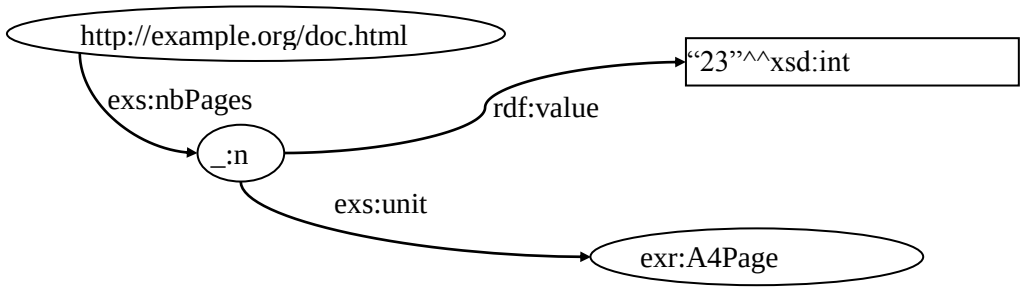

**RDF Graph**

### *2.3.3. Relaxing the binary graph constraint for n-ary relations*

RDF also provides primitives to structure the value of a property e.g., to give a value with its unit, its precision, a timestamp, etc. Structured values are usually represented by a blank node and the property `rdf:value` links it to the actual value. For instance to say that a document is 23-page long in A4 format:

(http://example.org/doc.html , rdf:type , exs:Report)
(http://example.org/doc.html , exs:nbPages , _:n)
(_:n, rdf:value , "23"^^xsd:int)
(_:n, exs:unit , exr:A4Page)

**RDF Triples**




**RDF Graph**

More generally speaking, RDF is a binary relation model. N-ary relations have to be decomposed and design patterns for this modeling problem and others are proposed in the semantic web best practices and deployment working group at W3C [32].

### *2.3.4. Semantic-less reification*

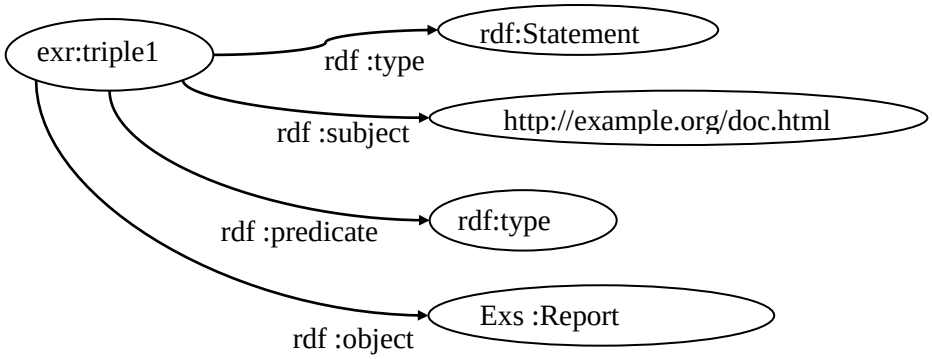
Finally, RDF provides primitives to explicitly represent and describe the triples without asserting them, a reification mechanism to allow statements about statements. RDF specification does not assign a normative formal semantics to the reification vocabulary and it won't be developed here.

(http://example.org/doc.html , rdf:type , exs:Report)

**RDF Triple**

(exr:triple1, rdf:type, rdf:Statement )
(exr:triple1, rdf:subject, http://example.org/doc.html)
(exr:triple1, rdf:predicate, rdf:type)
(exr:triple1, rdf:object, exs:Report)

**Reified Triple**




**RDF Graph**

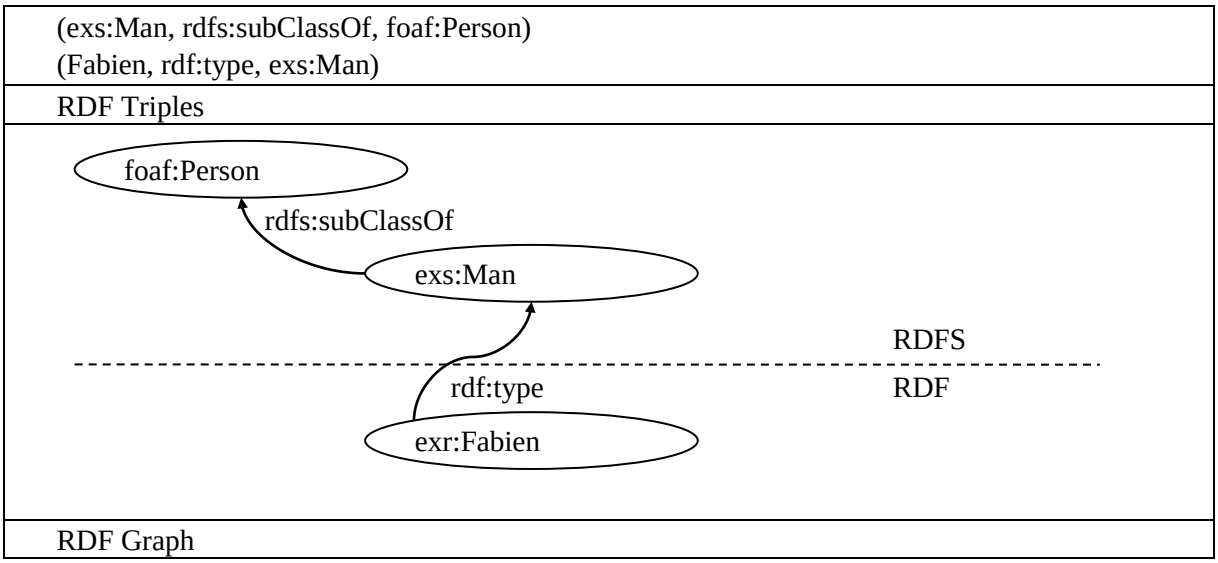
So RDF provides constructs to write reification quads but in RDF asserting the reification is not the same as asserting the original statement, and neither implies the other. Moreover reification expands the initial triple into a total of five triples (a triple plus a reification quad) and the link between the initial triple and its reification quad is not maintained.

## *2.4. Lightweight Ontology Formalization in RDFS*

RDFS stands for RDF schema and is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources [6]. It is a lightweight language to declare and describe the resource types (called classes) and resource relationship and attribute types (called properties). RDFS allows one to name and define vocabularies used in labeling RDF graphs: naming the classes of existing resources; naming relation types existing between instances of these classes and giving their signatures, *i.e.*, the type of resources they connect. RDFS defines inferences to be applied using these hierarchies of types and the signatures of properties. Providing a URI for types, RDFS allows one to declare the taxonomic skeleton of an ontology in a universal language, with universal identifiers and semantics.

### *2.4.1. Taxonomical skeleton of resource classes*

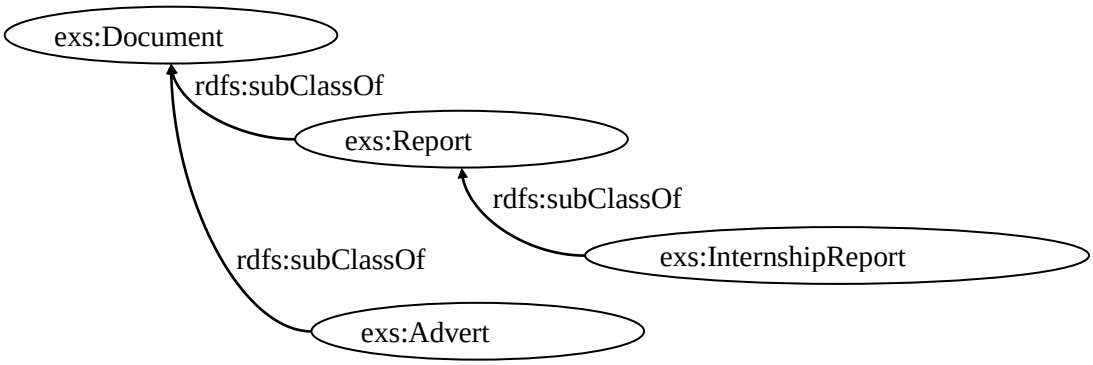
RDFS definitions factorize some of the information about the RDF data so that it is no longer needed to repeat that information. The information is no longer explicitly said in the data but can be derived through inferences. For instance by saying that the class `Man` (a set of resources) is a subclass (subset) of the class `Person` (another set of resources) it's no longer required to say that `Fabien` is a `Person` and a `Man`: this can be derived from the fact `Fabien` is a `Man` and the fact `Man` is a subclass of `Person` that `Fabien` is also a `Person`.




All things described in RDF are of type `rdfs:Resource`. It is the class of all things in RDF. The resource `rdfs:Resource` is of type `rdfs:Class` and every other class in RDF is a subclass of `rdfs:Resource`. The class `rdfs:Class` is the type of all classes in RDF and is itself a resource of type `rdfs:Class`. The class `rdfs:Literal` is the class of literal values such as strings and integers, is a subclass of `rdfs:Resource` and is an instance of `rdfs:Class`. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed. A typed literal is an instance of a datatype class. `rdfs:Datatype` is the class of datatypes and is both an instance of and a subclass of `rdfs:Class`. Each instance of `rdfs:Datatype` is a sub-class of `rdfs:Literal`. The class `rdf:XMLLiteral` is the class of XML literal values, is an instance of `rdfs:Datatype` and a sub-class of `rdfs:Literal`.

Properties, used to label the arcs of the RDF graphs, are first class citizens like classes. The class `rdf:Property` is the class of all RDF properties. Classes and properties are organized in two hierarchies allowing multiple inheritances. The hierarchy of classes is given by links of type `rdfs:subClassOf` between the classes. The hierarchy of properties is given by links of type

`rdfs:subPropertyOf` between the properties. In the example below, an `InternshipReport` is a subclass of `Report` which is itself a subclass of `Document`.

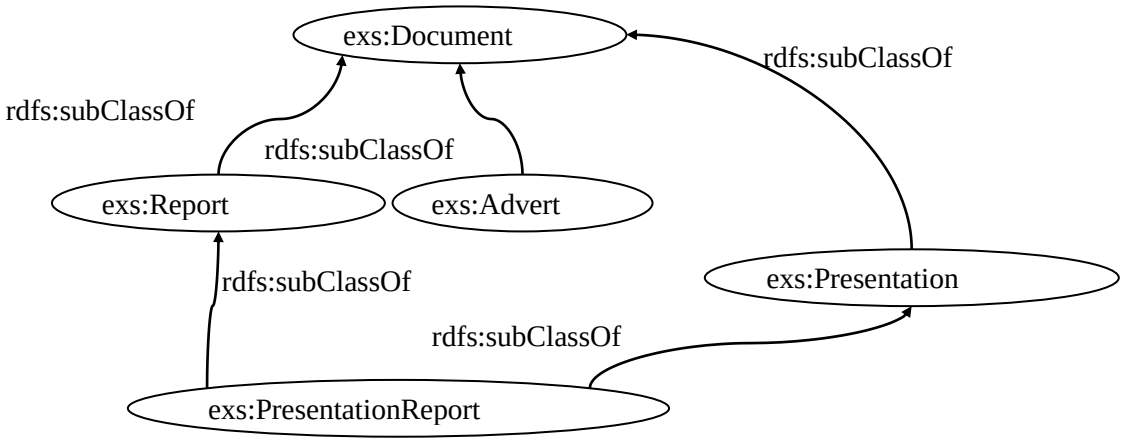
| |
|---|
| (exs:Report, rdfs:subClassOf, exs:Document)<br>(exs:Advert, rdfs:subClassOf, exs:Document)<br>(exs:InternshipReport, rdfs:subClassOf, exs:Report) |
| RDF Triples |
|  |
| RDF Graph |

As introduced earlier `rdf:type` is the property used to state that a resource is an instance of a class. All the instances of a class are instances of its super-classes *i.e.* instances of a class are instances of classes it is a subclass of. In addition, `rdfs:subClassOf` property is transitive *i.e.* all super classes of a class are also super classes of its sub-classes. The type of a resource propagates through the hierarchy defined by `rdfs:subClassOf`. In the previous example `InternshipReport` is also a subclass of `Document` and every instance of `InternshipReport` is also an instance of `Report` and of `Document`.

The property `rdfs:subClassOf` allows multiple inheritance. In other word not only can a class have several children, it can also have several parents. Having several children can be seen as a certain kind of union (the parent class includes the union of its children). Having several parents can be seen as a certain kind of intersection (the child class is included in the intersection of its parents).

In the following example `Document` has several children and can be seen as including the union of `Report`, `Advert` and `Presentation`. `PresentationReport` has several parents which are `Report` and `Presentation` and can be seen included in the intersection of `Report` and `Presentation`.

| |
|---|
| (exs:Report, rdfs:subClassOf, exs:Document)<br>(exs:Advert, rdfs:subClassOf, exs:Document)<br>(exs:Presentation, rdfs:subClassOf, exs:Document)<br>(exs:PresnetationReport, rdfs:subClassOf, exs:Report)<br>(exs:PresnetationReport, rdfs:subClassOf, exs:Presentation) |
| RDF Triples |



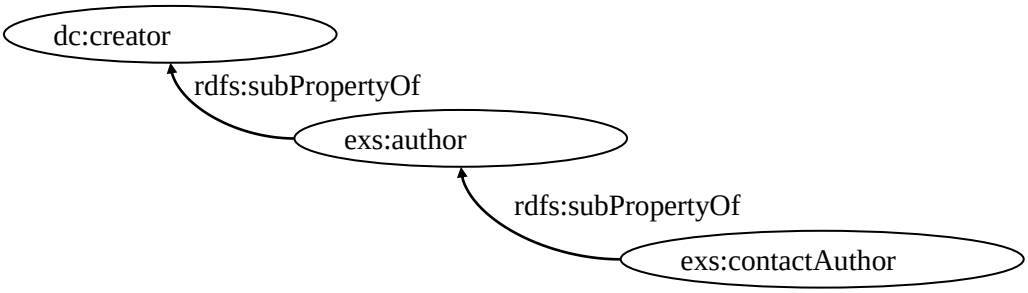

RDF Graph

### *2.4.2. Taxonomical skeleton of resource relations*

In the same manner, the property `author` is a sub-type of `creator` and inherits its signature.

(exs:author, rdfs:subPropertyOf, dc:creator)
(exs:contactAuthor, rdfs:subPropertyOf, exs:author)

RDF Triples



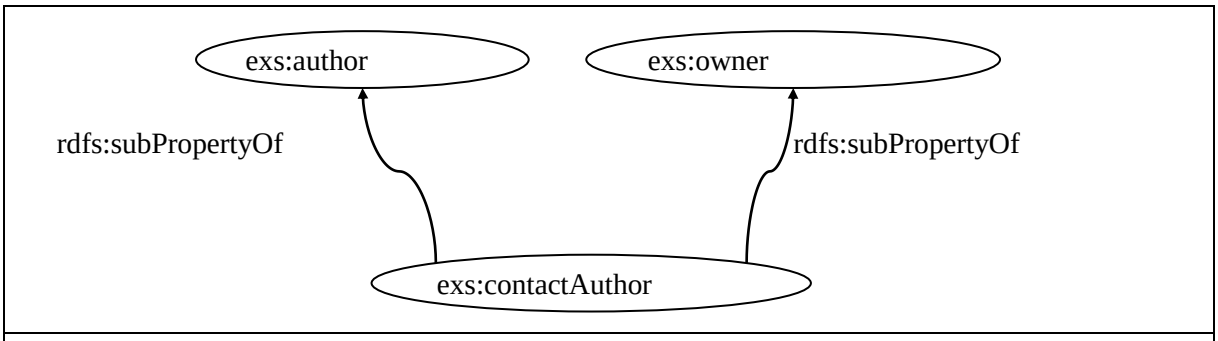

RDF Graph

Likewise, all resources related by one property are also related by its super-properties *i.e.* properties it is a sub-property of. In addition, `rdfs:subPropertyOf` property is transitive *i.e.* all super properties of a property are also super properties of its sub-properties. Relationships propagate through hierarchy defined by `rdfs:subPropertyOf`. In the previous example `contactAuthor` is also a sub-property of `creator` and when a relation `contactAuthor` holds between two resources then the relations `author` and `creator` also hold.

The property `rdfs:subPropertyOf` allows multiple inheritance. As it is the case for classes not only can a property have several children, it can also have several parents. Using multiple inheritance it could be said that the property `contactAuthor` means that someone is the `author` and the `owner` of the document.

(exs:contactAuthor, rdfs:subPropertyOf, exs:author)
(exs:contactAuthor, rdfs:subPropertyOf, exs:owner)

RDF Triples



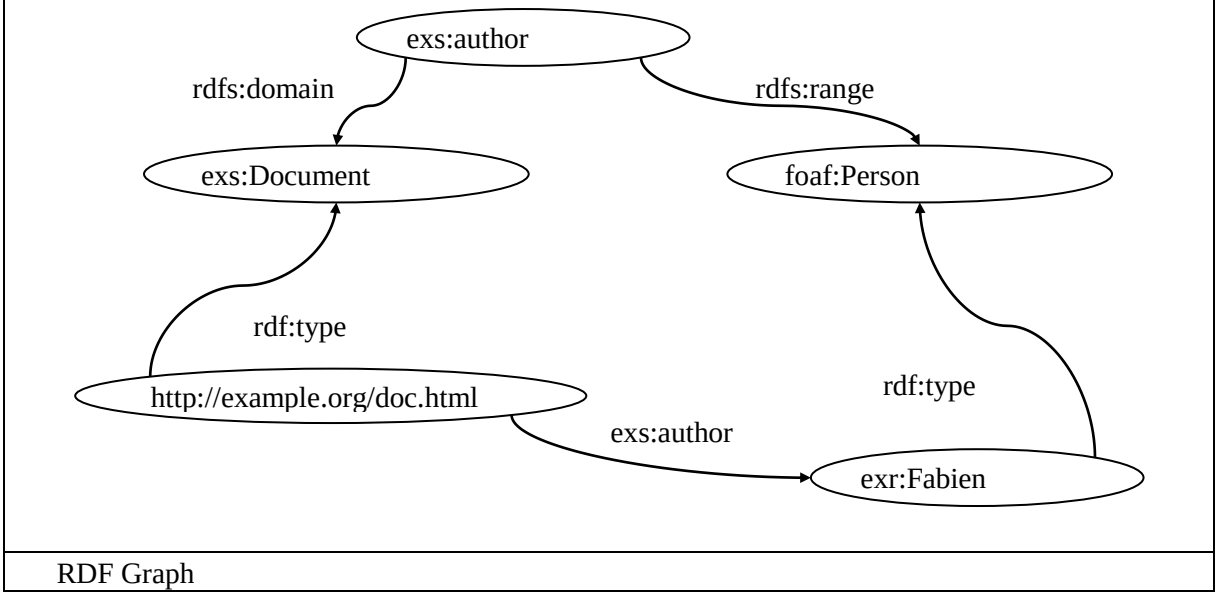

RDF Graph

The signature of a property gives the classes of resources that are linked by this property. The property `rdfs:domain` is used to state that any resource that has a given property is an instance of a given class. The property `rdfs:range` is used to state that the values of a property are instances of a given classes. When several domains or ranges are given, the instances belong to all the given classes. The signature of properties allows one to type data through their usage: every time a property is used the resources it links will be typed using its domains and its ranges. The terms domain and range were borrowed from mathematics where the domain of a function is the set a values for which it is defined and the range is the set of values it can take. In RDFS a property relates resources belonging to its domain to resources belonging to its range.

The property author has a signature saying it links a `Document` to a `Person`. The statement "doc.html has author of Fabien" implies that `doc.html` is an instance of `Document` and `Fabien` is an instance of `Person`.

(exs:author, rdfs:range, foaf:Person)
(exs:author, rdfs:domain, exs:Document)
(http://example.org/doc.html , exs:author, exr:Fabien)

RDF Triples




RDF Graph

### *2.4.3. The meta-ontology RDF Schema*

RDFS provides primitives to declare lightweight ontologies. The degree of formality and expressivity of the language used to describe them forms a continuum of ontology kinds as shown in

Figure 4, starting from terms and web directories, and continuing to rigorously formalized logical theories. For a long time, lightweight ontologies were not formally defined but were referred to by examples. For instance, they were referred to as terms, as controlled vocabularies, as thesauri, and as web directories. As it can be observed, informal lightweight ontologies largely cover the spectrum of informal ontology kinds shown in Figure 4. In some other approaches, lightweight ontologies are formal ontologies which use a computationally inexpensive logic language.

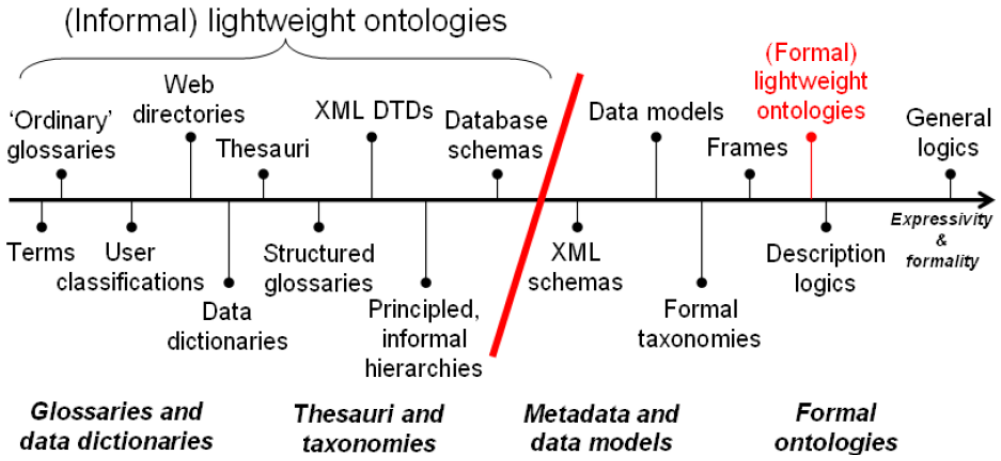



Fig. 4. Lightweight ontologies [from:Fausto Giunchiglia and Ilya Zaihrayeu: LIGHTWEIGHT ONTOLOGIES - October 2007 – Technical Report DIT-07-071]

The following tables (Table 5 and Table 6) present an overview of the vocabulary of provided to represent lightweight ontologies, drawing together vocabulary originally defined in the RDF Model and Syntax specification with classes and properties that originate with RDF Schema.

Table 5. RDF Classes

| Class name | Comment |
|---|---|
| rdfs:Resource | The class of resource, everything. |
| rdfs:Literal | The class of literal values, e.g. textual strings and integers. |
| rdf:XMLLiteral | The class of XML literals values. |
| rdfs:Class | The class of classes. |
| rdf:Property | The class of RDF properties. |
| rdfs:Datatype | The class of RDF datatypes. |
| rdf:Statement | The class of RDF statements. |
| rdf:Bag | The class of unordered containers. |
| rdf:Seq | The class of ordered containers. |
| rdf:Alt | The class of containers of alternatives. |
| rdfs:Container | The class of RDF containers. |
| rdfs:ContainerMembershipProperty | The class of container membership properties, `rdf:_1`, `rdf:_2`, ..., all of which are sub-properties of 'member'. |
| rdf:List | The class of RDF Lists. |

Table 6. RDF Properties

| name | comment | domain | Range |
|---|---|---|---|
| rdf:type | The subject is an instance of the class given as object. | rdfs:Resource | rdfs:Class |
| rdfs:subClassOf | The subject is a subclass of the class given as | rdfs:Class | rdfs:Class |

| | | | |
|---|---|---|---|
| | object. | | |
| rdfs:subPropertyOf | The subject is a subproperty of the property given as object. | rdf:Property | rdf:Property |
| rdfs:domain | A domain of the subject property. | rdf:Property | rdfs:Class |
| rdfs:range | A range of the subject property. | rdf:Property | rdfs:Class |
| rdfs:label | A human-readable name for the subject. | rdfs:Resource | rdfs:Literal |
| rdfs:comment | A description of the subject resource. | rdfs:Resource | rdfs:Literal |
| rdfs:member | A member of the subject resource. | rdfs:Resource | rdfs:Resource |
| rdf:first | The first item in the subject RDF list. | rdf:List | rdfs:Resource |
| rdf:rest | The rest of the subject RDF list after the first item. | rdf:List | rdf:List |
| rdfs:seeAlso | Further information about the subject resource. | rdfs:Resource | rdfs:Resource |
| rdfs:isDefinedBy | The definition of the subject resource. | rdfs:Resource | rdfs:Resource |
| rdf:value | Idiomatic property used for structured values | rdfs:Resource | rdfs:Resource |
| rdf:subject | The subject of the subject RDF statement. | rdf:Statement | rdfs:Resource |
| rdf:predicate | The predicate of the subject RDF statement. | rdf:Statement | rdfs:Resource |
| rdf:object | The object of the subject RDF statement. | rdf:Statement | rdfs:Resource |

RDFS is about sets and subsets *i.e.* the semantics of RDFS is based on sets and set operators (union intersection and inclusion). Its seen that that RDF provides the membership declaration property (`rdf:type`) that allows one to capture in a graph structure which resource belong to which class (a kind of set) and which couple of resources belong to which relation (another kind of set). RDFS provides the vocabulary to describe the relations between these sets: relations between classes (this class is included in this other class), relations between properties (this property is included in this other property) and between classes and properties (this property links individuals from this class with individuals from this other class). Properties and their definitions are really important since a lot of the semantics are captured in the way resources are linked.

Figure 3 shows the relationship in RDF(S) vocabulary. A broken curved line stands for `rdf:type` relation, and a solid straight line stands for `rdfs:subClassOf` relation. `rdfs:Resource` is a superclass of all other classes, and `rdfs:Class` is a class of all classes, including `rdfs:Class` itself. A class of classes is called *metaclass* in CLOS(Common Lisp Object System). So, `rdfs:Class` and `rdfs:Datatype` in RDFS vocabulary are metaclasses in CLOS.

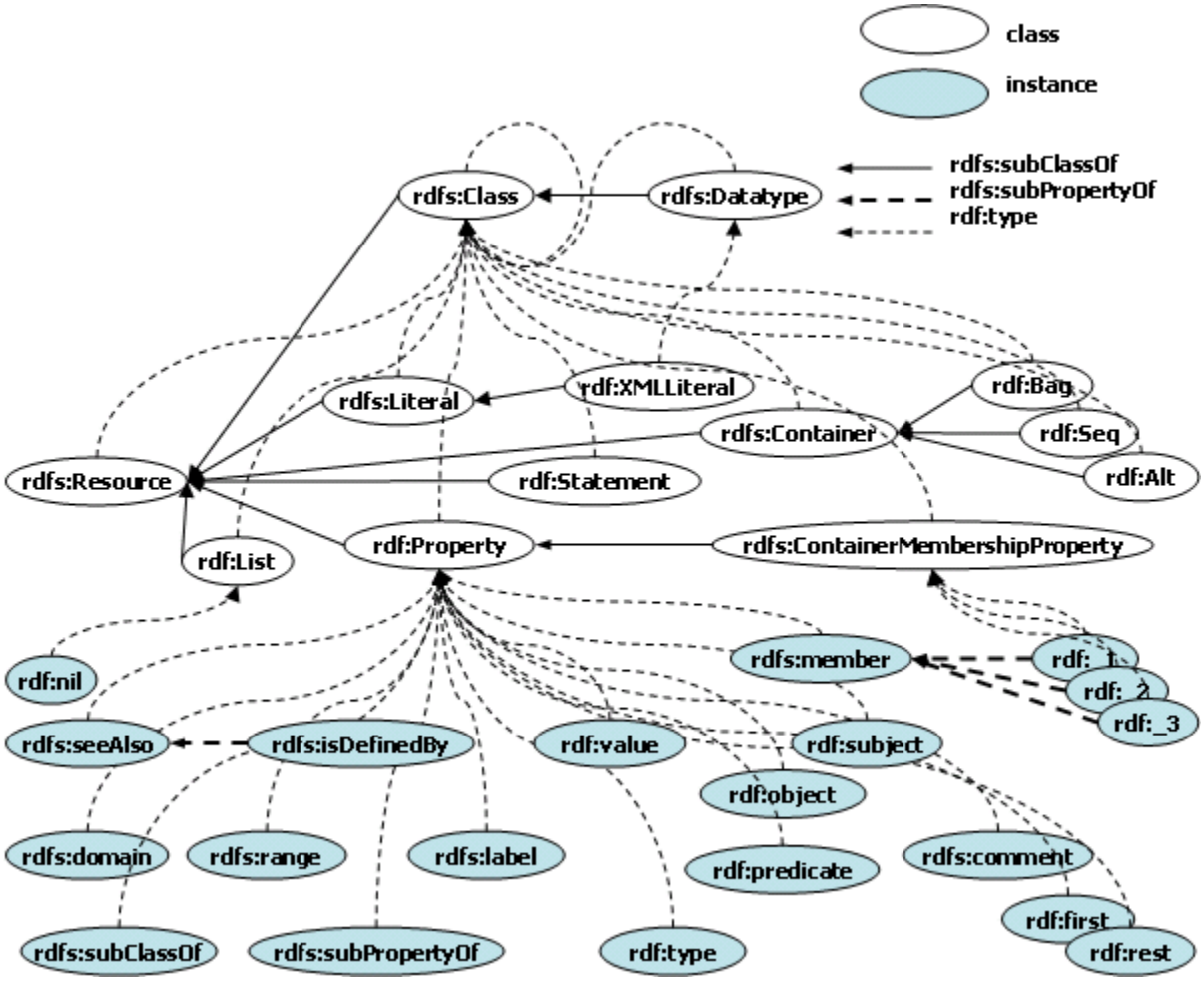


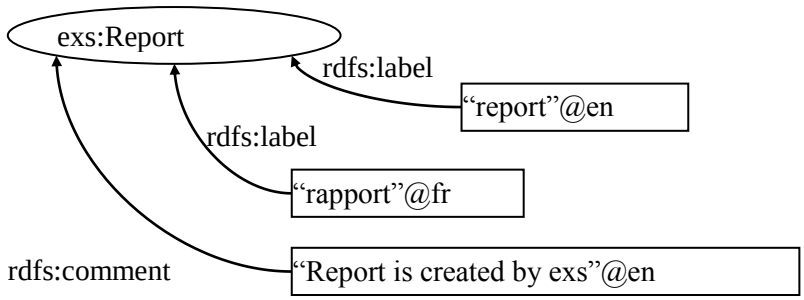

Fig. 3. RDFS Hierarchy Graph [24]

The `rdfs:Container` class is a super-class of the RDF Container classes which are `rdf:Bag`, `rdf:Seq` and `rdf:Alt` and these were explained earlier. The `rdfs:ContainerMembershipProperty` class has as for instances the properties `rdf:_1`, `rdf:_2`, `rdf:_3` ..., that are used to state that a resource is a member of a container. `rdfs:ContainerMembershipProperty` is a sub-property of `rdf:Property`. Each instance of `rdfs:ContainerMembershipProperty` is an `rdfs:subPropertyOf` the `rdfs:member` property. `rdfs:member` is an instance of `rdf:Property` that is a super-property of all the container membership properties. Each container membership property has an `rdfs:subPropertyOf` relationship to the property `rdfs:member`. The `rdfs:domain` of `rdfs:member` is `rdfs:Resource`. The `rdfs:range` of `rdfs:member` is `rdfs:Resource`.

`rdfs:seeAlso` is to indicate a resource that might provide additional information about the subject resource. `rdfs:isDefinedBy` is to indicate an RDF vocabulary in which a resource is described. The property `rdfs:isDefinedBy` is a subproperty of `rdfs:seeAlso`, and indicates a resource defining the subject resource. As with `rdf:seeAlso`, this property can be applied to any instance of `rdfs:Resource` and may have as its value any `rdfs:Resource`. The most common anticipated usage is to identify an RDF resource. Although XML namespace declarations will typically provide the URI where RDF vocabulary resources are defined, there are cases where additional information is required. For example, constructs such as `<rdfs:subPropertyOf rdf:resource = "http://purl.org/dc/elements/1.0/Creator"/>` do not indicate the URI of the schema that includes the vocabulary item Creator (i.e., http://purl.org/dc/elements/1.0/). In such cases, the `rdfs:isDefinedBy` property can be used to explicitly represent that information.

This approach will also work when the URIs of the namespace and its components have no obvious relationship, as would be the case if they were identified using schemes such as GUIDs or MD-5 hashes[http://www.w3.org/TR/rdf-schema/].

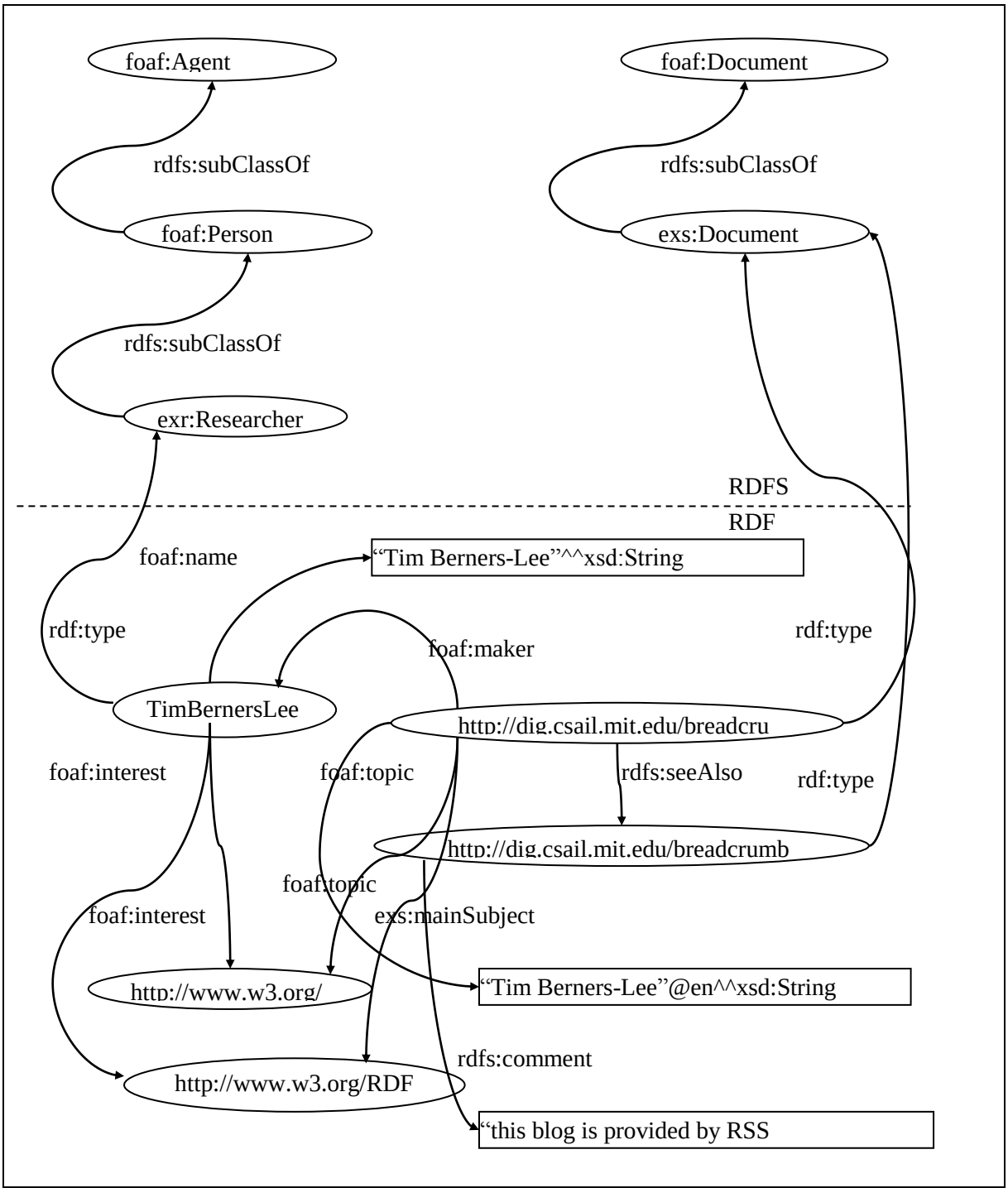
Finally, RDFS defines two properties to provide human-readable information about resources possibly in different natural languages and in particular to document classes and properties. The property `rdfs:label` is used to provide one or more names of a resource. The property `rdfs:comment` is used to provide one or more descriptions of a resource in natural language. The example below provides two labels for the class `exs:Report` one in English and one in French; this feature helps supporting internationalization.

| (exs:Report, rdfs:label, "report"@en)<br>(exs:Report, rdfs:label, "rapport"@fr)<br>(exs:Report, rdfs:comment, “Report is created by exs”@en) |
|---|
| RDF Triples |
|  |
| RDF Graph |

RDFS was defined in RDF *i.e.* RDFS itself and the schemas defined RDFS are written using RDF graphs and RDF triples. This is a very important feature which has tremendous advantages for instance the reuse of every RDF tools to manipulate the schemas and in particular the ability to use the SPARQL query language to query both the data and their schemas.

The following example shows how to use RDFS in RDF. The italic types in left of this example represent that the resources in RDF are the instances of classes in RDFS. For instance, Tim Berners-Lee is the instance of Researcher.

| (exs:Researcher, rdfs:subClassOf, foaf:Person)<br>(exr:TimBernersLee, rdf:type, exr:Researcher)<br>(exr:TimBernersLee, foaf:name, “Tim Berners-Lee”)<br>(exr:TimBernersLee, foaf:interest, http://www.w3.org/2001/sw/)<br>(exr:TimBernersLee, foaf:interest, http://www.w3.org/RDF/)<br>(exs:Document, rdfs:subClassOf, foaf:Document)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, rdf:type, exs:Document)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, exs:mainSubject, “Tim Berners-Lee”@en)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, rdfs:seeAlso, http://dig.csail.mit.edu/breadcrumbs/blog/feed/4)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/feed/4, rdf:type, exs:Document)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/feed/4, rdfs:comment, “this blog is provided by RSS channel ”@en)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, rdfs:maker, exr:TimBernersLee)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, foaf:topic, http://www.w3.org/2001/sw/)<br>(http://dig.csail.mit.edu/breadcrumbs/blog/4, foaf:topic, http://www.w3.org/RDF/) |
|---|
| RDF Triples |




RDF Graph

## *2.5. RDF/S Semantics*

The following sections deal with the semantics of RDF and RDFS. The idea behind that is to put sentences, the syntactical expressions of logics, in relation to interpretations. Such interpretations are all possible manifestations of the real world. It is not necessary that those interpretations are compliant to the real world, however in order to guarantee correctness one has to use mathematical structures for interpretations. In the following figure we can see how the mentioned sentences and the interpretations relate among each other:

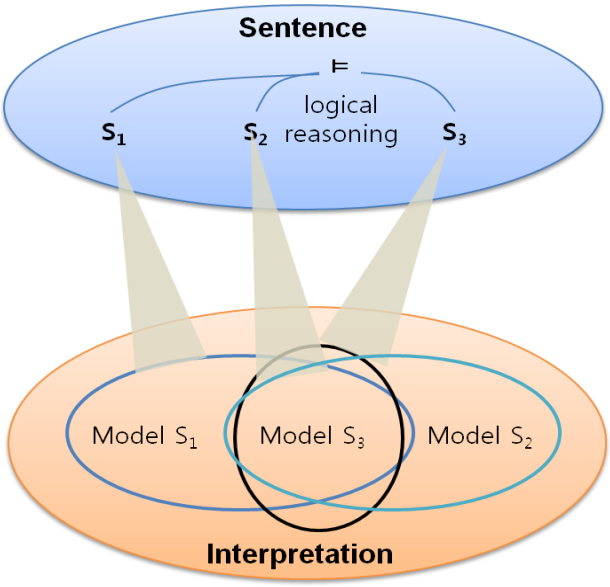



Figure @@@ Definition of logical reasoning relation over Model

In order to define the semantics of RDF and RDFS we will start with simple interpretations. Then we describe additional criteria which have to be fulfilled by a simple interpretation in order to be usable as RDF interpretation. Adding further criteria to a RDF interpretation will enable us to use RDFS interpretations. Finally we will explain how external data types fit into RDFS interpretations.

### *2.5.1. Simple interpretation*

For simple interpretations, the first thing we need is a vocabulary. A vocabulary **V** is a set of URIs and literals. Furthermore we need resources and properties which define how the resources are related to each other. For the resources we use the set **IR** and for the properties we use the set **IP**. In order to state which resources are related to each other by using a certain properties we use the function **I_EXT**. With these components we can say that a simple interpretation **I** for a vocabulary **V** consists of the following:

- **IR, a nonempty set of resources (domain, universe of I),**
- **IP, a set of properties of I,**
- **I_EXT, a function mapping a property to a set of resources,**
- **I_S, a function mapping URIs from V to the union of the sets IR and IP,**
- **I_L, a function mapping typed literals from V to the set IR of resources and**
- **LV, a specific subset of IR which (at least) contains all untyped literals from V.**

The interpretation function **I** maps all literals and URIs to resources and properties. The following figure shows how the just described interpretation works:

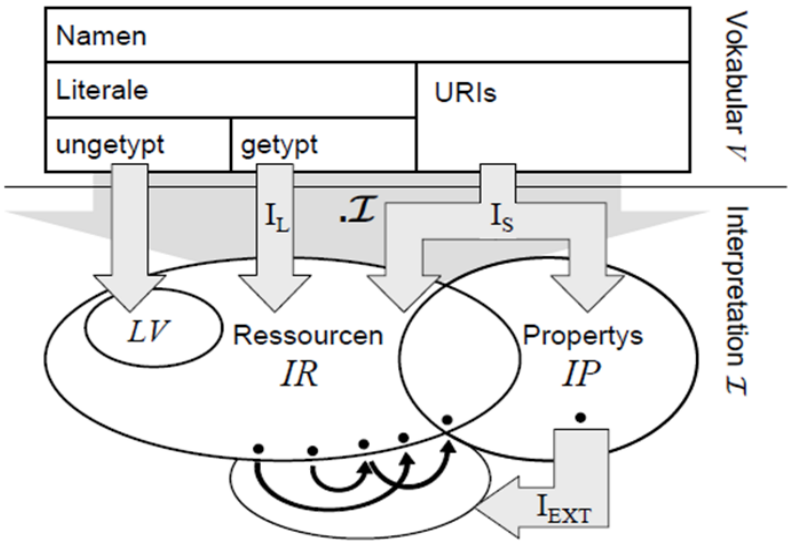



Figure @@@ Schematic representation of simple inprementation

The following figure shows the criterion for the validity of a triple regarding a certain interpretation:

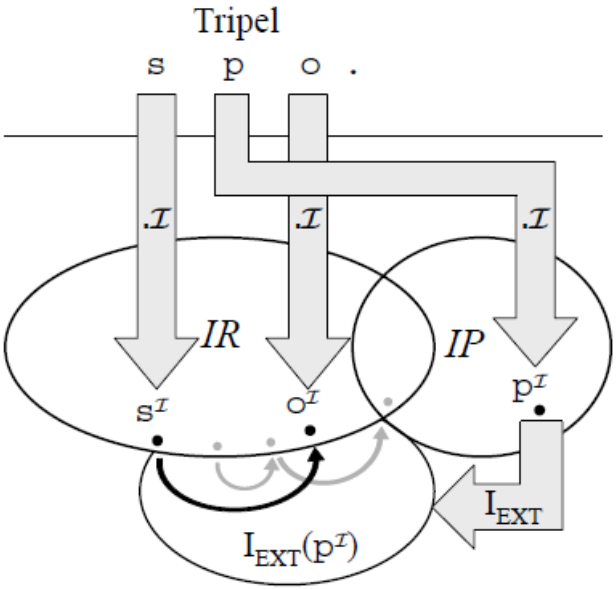



Figure @@@ Criteria for the effectiveness of triples by means of interpretation

### *2.5.2. RDF Interpretation*

In simple interpretations all URIs occurring in a vocabulary are treated the same regardless namespace or meaning. In order to enable RDF interpretations we have to set up new criteria. Therefore we define the RDF vocabulary **V_RDF** which consists of the following URIs:

```
rdf:type rdf:Property rdf:XMLLiteral rdf:nil
rdf:List rdf:Statement rdf:subject rdf:predicate rdf:object
rdf:first rdf:rest rdf:Seq rdf:Bag rdf:Alt
rdf:_1 rdf:_2 ...
```

A RDF interpretation for a vocabulary **V** is a simple interpretation for that vocabulary with additional requirements:

- **x is a property if it is related to a resource via the rdf:type relation,**
- **in case of "s"^^rdf:XMLLiteral is contained in V and s is a well-formed XML literal then**
    - **I_L ("s"^^rdf:XMLLiteral) is the XML value of s;**

    - **I_L ("s"^^rdf:XMLLiteral) is element of LV;**
    - **the pair of I_L ("s"^^rdf:XMLLiteral) and rdf:XMLLiteral is element of I_EXT(rdf:type)**
- **in case of "s"^^rdf:XMLLiteral is contained in V and s is not a well-formed XML literal then**
    - **I_L ("s"^^rdf:XMLLiteral) is not element of LV;**
    - **the pair of I_L ("s"^^rdf:XMLLiteral) and rdf:XMLLiteral is not element of I_EXT(rdf:type)**

Furthermore the following axiomatic RDF triples must hold in order to enable RDF interpretations:

```
rdf:type        rdf:type    rdf:Property .
rdf:subject     rdf:type    rdf:Property .
rdf:predicate   rdf:type    rdf:Property .
rdf:object      rdf:type    rdf:Property .
rdf:first       rdf:type    rdf:Property .
rdf:rest        rdf:type    rdf:Property .
rdf:value       rdf:type    rdf:Property .
rdf:_1          rdf:type    rdf:Property .
rdf:_2          rdf:type    rdf:Property .
...
rdf:nil         rdf:type    rdf:List .
```

### *2.5.3. RDFS Interpretation*

Since RDFS enriches RDF with several new constructs we also have to introduce a new function **I_CEXT** in order to enable RDFS interpretations.
A RDFS interpretation for a vocabulary **V** is a RDF interpretation for that vocabulary **V** which additionally has to be consistent to the following criteria:

- **Every resource is of type rdfs:Resource,**
- **every untyped and every typed literal is of type rdfs:Literal,**
- **in case of x being related to y via the property rdfs:domain and the resource u being related to v via that property, then v is of type y,**
- **in case of x being related to y via the property rdfs:range and the resource u being related to v via that property, then v is of type y,**
- **rdfs:SubPropertyOf is transitive and reflexive in IP,**
- **in case of x being related to y via rdfs:SubPropertyOf then both are properties and every pair of resources contained in the extension of x is also contained in the extension of y,**
- **in case of x being an identifier for a class then it has to be a subclass of the class of all resources,**
- **in case of x being related to y via rdfs:SubClassOf then both are classes and the extension of x is a subset of the extension of y,**
- **rdfs:SubClassOf is transitive and reflexive in IC,**
- **in case of x being a property of type rdfs:ContainerMembershipProperty, x relates to the rdfs:member property via the rdfs:SubPropertyOf relation,**
- **in case of x being an element of the class rdfs:Datatype then it has to be a subclass of the class of all literal values.**

Furthermore we have to define a set of triples that must hold for a RDF interpretation in order to serve as a RDFS interpretation:

```
rdf:type            rdfs:domain        rdfs:Resource .
rdfs:domain         rdfs:domain        rdf:Property .
rdfs:range          rdfs:domain        rdf:Property .
rdfs:subPropertyOf  rdfs:domain        rdf:Property .
rdfs:subClassOf     rdfs:domain        rdfs:Class .
rdf:subject         rdfs:domain        rdf:Statement .
rdf:predicate       rdfs:domain        rdf:Statement .
rdf:object          rdfs:domain        rdf:Statement .
rdfs:member         rdfs:domain        rdfs:Resource .
rdf:first           rdfs:domain        rdf:List .
rdf:rest            rdfs:domain        rdf:List .
rdfs:seeAlso        rdfs:domain        rdfs:Resource .
rdfs:isDefinedBy    rdfs:domain        rdfs:Resource .
rdfs:comment        rdfs:domain        rdfs:Resource .
rdfs:label          rdfs:domain        rdfs:Resource .
rdf:value           rdfs:domain        rdfs:Resource .

rdf:type            rdfs:range         rdfs:Class .
rdfs:domain         rdfs:range         rdfs:Class .
rdfs:range          rdfs:range         rdfs:Class .
rdfs:subPropertyOf  rdfs:range         rdf:Property .
rdfs:subClassOf     rdfs:range         rdfs:Class .
rdf:subject         rdfs:range         rdfs:Resource .
rdf:predicate       rdfs:range         rdfs:Resource .
rdf:object          rdfs:range         rdfs:Resource .
rdfs:member         rdfs:range         rdfs:Resource .
rdf:first           rdfs:range         rdfs:Resource .
rdf:rest            rdfs:range         rdf:List .
rdfs:seeAlso        rdfs:range         rdfs:Resource .
rdfs:isDefinedBy    rdfs:range         rdfs:Resource .
rdfs:comment        rdfs:range         rdfs:Literal .
rdfs:label          rdfs:range         rdfs:Literal .
rdf:value           rdfs:range         rdfs:Resource .
```

```
rdfs:ContainerMembershipProperty
                    rdfs:subClassOf     rdf:Property .
rdf:Alt             rdfs:subClassOf     rdfs:Container .
rdf:Bag             rdfs:subClassOf     rdfs:Container .
rdf:Seq             rdfs:subClassOf     rdfs:Container .

rdfs:isDefinedBy    rdfs:subPropertyOf  rdfs:seeAlso .

rdf:XMLLiteral      rdf:type            rdfs:Datatype .
rdf:XMLLiteral      rdfs:subClassOf     rdfs:Literal .
rdfs:Datatype       rdfs:subClassOf     rdfs:Class .

rdf:_1              rdf:type
                       rdfs:ContainerMembershipProperty .
rdf:_1              rdfs:domain         rdfs:Resource .
rdf:_1              rdfs:range          rdfs:Resource .
rdf:_2              rdf:type
                       rdfs:ContainerMembershipProperty .
rdf:_2              rdfs:domain         rdfs:Resource .

 rdf:_2              rdfs:range          rdfs:Resource .
```

### *2.5.4. Datatype interpretation*

In RDFS we solely have one predefined data type which is **rdf:XMLLiteral**. Its semantic properties are covered by the RDFS interpretations described previously. Yet it is possible to use externally defined data types. Therefore we use URIs for these data types in the vocabulary **V** in order to be able to formulate expressions about these data types. For the corresponding interpretation we define another function **D** which maps such data type-URIs to their data types.

### *2.5.5. Deductions*

The previous section described the semantics of RDF and RDFS which can be used to state how one RDF and RDFS graph can be deduced from another RDF and RDFS graph. Such semantics however are not very useful when it comes to automatic decision making regarding whether one graph can be deduced from another one. There we would have to take all interpretations into account to decide the just mentioned question. Here the problem is that there are always infinitely many interpretations with infinitely many elements each. Thus it is not possible to take them all into account.
To solve this issue we have to stay at the syntactical level without using interpretations. For that we can use deduction rules of the following form:

$$\frac{s_1 \quad \cdots \quad s_n}{s}$$

This rule states that whenever **s1** to **sn** are elements of the set of true expressions, then also **s** is element of that set.
In the following we use deduction rules in order to enable simple deduction, RDF deduction, RDFS deduction and deduction for externally defined data types.

***Simple deduction***

The rules for simple deduction are rather simple in their characteristics. Due to the fact that all occurring URIs are treated the same we can answer the question, whether one graph can be deduced from another one, by using simple structural facts which are covered in the following deduction rules:

$$\frac{u\ \ a\ \ x\ \ .}{u\ \ a\ \ \_{:}n\ \ .}\ \text{se1}$$

$$\frac{u\ \ a\ \ x\ \ .}{\_{:}n\ \ a\ \ x\ \ .}\ \text{se2}$$

Here one has to ensure that **_:n** had not been attached to other URIs, knots or literals by applying one of the rules **se1**, **se2**. Expressed in words we can say: Replace some occurrences of a specific URI of a specific empty knot or specific literal of a graph by a new empty knot. It is not allowed that such a knot has occurred in the graph before.

***RDF Deduction***

In contrast to simple deduction, RDF deduction needs certain URIs with a specific meaning. Therefore we have to extend the set of deduction rules. As a first step we use rules that do not need any preconditions which is why they can be applied at any point in time:

$$\frac{}{u\ \ a\ \ x}\ \text{rdfax}$$

Furthermore we need to alter rule **se1**, yet we use the same restrictions as for **se1**:

$$\frac{u\ \ a\ \ l\ \ .}{u\ \ a\ \ \_{:}n\ \ .}\ \text{lg}$$

To ensure that each URI, which is used as predicate in a graph, is assigned to the type **rdf:Property**, we use the following rule:

$$\frac{u\ \ a\ \ y\ \ .}{a\ \ \texttt{rdf:type}\ \ \texttt{rdf:Property}\ \ .}\ \text{rdf1}$$

$$\frac{u\ \ a\ \ l\ \ .}{\_{:}n\ \ \texttt{rdf:type}\ \ \texttt{rdf:XMLLiteral}}\ \text{rdf2}$$

In the previous rule **l** is a XML-literal and **_:n** defines an empty knot which is assigned to **l** by the rule **lg**.

These rules together with the simple deduction rules of the previous section enable RDF deduction.

For enabling RDF deduction we have to introduce a bunch of new deduction rules:

***RDFS Deductions***

Similar to the rule **rdfax** from the RDF deduction we now have a rule **rdfsax** to deduce from any axiomatic triple:

$$\frac{}{u\ \ a\ \ x}\ \text{rdfsax}$$

Dealing with literals:

$$\frac{u\ \ a\ \ \_{:}n\ \ .}{u\ \ a\ \ l\ \ .}\ \text{gl}$$

In this rule, **_:n** defines an empty knot assigned to **l** by applying the rule **lg**.

$$\frac{u\ \ a\ \ l\ .}{\_{:}n\ \ \texttt{rdf:type}\ \ \texttt{rdfs:Literal}\ \ .}\ \text{rdfs1}$$

In the rule above, **l** is an untyped literal and **_:n** is again an empty knot assigned to **l** by applying the rule **lg**. These two rules enable to deduce existential propositions from literals.

Property signature : **rdf:domain** is used to state that a property can be assigned only to elements of a certain class. This can be ensured by using the rule **rdfs2**:

*2.5.6.* $\dfrac{a\ \texttt{rdfs:domain}\ x\ .\quad u\ a\ y\ .}{u\ \texttt{rdf:type}\ x\ .}$ rdfs2

Similarly we use **rdfs3** to ensure that the values of a property belong to a certain class:

$$\frac{a\ \texttt{rdfs:range}\ x\ .\quad u\ a\ v\ .}{v\ \texttt{rdf:type}\ x\ .}\ \text{rdfs3}$$

Everything is a resource:
To ensure that each URI occurring in a triple can be recognized as an identifier for a resource we use the following rules:

$$\frac{u\ a\ x\ .}{u\ \texttt{rdf:type rdfs:Resource}\ .}\ \text{rdfs4a}$$

$$\frac{u\ a\ v\ .}{v\ \texttt{rdf:type rdfs:Resource}\ .}\ \text{rdfs4b}$$

Subproperties:
The following rules are used to guarantee that transitivity (**rdfs5**) and reflexivity (**rdfs6**) of **rdfs:subPropertyOf** are accessible:

$$\frac{u\ \texttt{rdfs:subPropertyOf}\ v\ .\quad v\ \texttt{rdfs:subPropertyOf}\ x\ .}{u\ \texttt{rdfs:subPropertyOf}\ x\ .}\ \text{rdfs5}$$

$$\frac{u\ \texttt{rdf:type rdf:Property}\ .}{u\ \texttt{rdfs:subPropertyOf}\ u\ .}\ \text{rdfs6}$$

**rdf7** states that the property relations between pairs of resources are also available via the superproperty:

$$\frac{a\ \texttt{rdfs:subPropertyOf}\ b\ .\quad u\ a\ y\ .}{u\ b\ y\ .}\ \text{rdfs7}$$

Subclasses:
**rdfs8** states that each class, identified by a class identifier, is a subclass of the class of all resources:

$$\frac{u\ \texttt{rdf:type rdfs:Class}\ .}{u\ \texttt{rdfs:subClassOf rdfs:Resource}\ .}\ \text{rdfs8}$$

The fact that a resource is contained in a class can also be carried on the superclass level:

$$\frac{u\ \texttt{rdfs:subClassOf}\ x\ .\quad v\ \texttt{rdf:type}\ u\ .}{v\ \texttt{rdf:type}\ x\ .}\ \text{rdfs9}$$

Finally, **rdfs11** allows to state that every class is a subclass of itself:

$$\frac{u\ \texttt{rdfs:subClassOf}\ v\ .\quad v\ \texttt{rdfs:subClassOf}\ x\ .}{u\ \texttt{rdfs:subClassOf}\ x\ .}\ \text{rdfs11}$$

Container:
**rdfs:member** is superproperty of all properties in **rdfs:ContainerMembershipProperty**:

$$\frac{u\ \texttt{rdf:type rdfs:ContainerMembershipProperty}\ .}{u\ \texttt{rdfs:subPropertyOf rdfs:member}\ .}\ \text{rdfs12}$$

Literals:
The last rule states that each resource defined as data type is a subclass of all literal values:

$$\frac{u\ \texttt{rdf:type rdfs:Datatype}\ .}{u\ \texttt{rdfs:subClassOf rdfs:Literal}\ .}\ \text{rdfs13}$$

### *2.5.7. Rules for data types*

All the previously mentioned deduction rules are correct and complete for RDFS interpretations. If one wants to add external data types this can be achieved by introducing them to the set of elements of

the class defined by **rdfs:datatype**, however it is not possible to fully characterize their behavior by just using RDFS graphs. Yet it is possible to express certain types of relations between data types which occur rather often.

$$\frac{d \text{ rdf:type rdfs:Datatype .} \quad u \ a"s"\hat{}\hat{}d \ .}{\_:n \text{ rdf:type } d \ .} \text{ rdfD1}$$

The rule above allows assuming that whenever a literal occurs then there exists a corresponding type of resource.
Another case which might occur rather often is when different data types overlap:

$$\frac{\begin{array}{l} d \text{ rdf:type rdfs:Datatype .} \\ e \text{ rdf:type rdfs:Datatype .} \\ u \ a \ "s"\hat{}\hat{}d \ . \end{array}}{u \ a \ "t"\hat{}\hat{}e \ .} \text{ rdfD2}$$

**rdfD2** states that every occurrence of a typed literal as object of a triple can be replaced by the other literal.

////

The meaning of RDF and RDFS vocabularies is represented with axioms and entailment rules. Let us see the RDF vocabularies at first. The major role of RDF is to define the class membership property, `rdf:type`, and the property class, `rdf:Property`. All properties defined in RDF are members of `rdf:Property` class. This is represented as the following RDF axioms which are always true. *n* of rdf:*_n* can be substituted by any positive integer.

| | |
|---|---|
| rdf:type rdf:type rdf:Property .<br>rdf:subject rdf:type rdf:Property .<br>rdf:predicate rdf:type rdf:Property .<br>rdf:object rdf:type rdf:Property .<br>rdf:first rdf:type rdf:Property . | rdf:rest rdf:type rdf:Property .<br>rdf:value rdf:type rdf:Property .<br>rdf:*_n* rdf:type rdf:Property . |

RDF vocabularies also have a formal meaning which entails some conclusions – *i.e.*, RDF statements – from a given RDF graph. This meaning is represented as the following two RDF entailment rules. The question marks (?) denote variables and *isXMLLiteral*() checks if its argument is a well-typed XML literal. rdf1 represents that a predicate is a property and rdf2 represents the data type of well-typed XML literal.

| Rule name | Conditions | Conclusions |
|---|---|---|
| rdf1 | (?x ?a ?y) | (?a rdf:type rdf:Property) |
| rdf2 | (?x ?a ?y) *isXMLLiteral*(?y) | (?y rdf:type rdf:XMLLiteral) |

Next, let us see the RDFS vocabularies. The major role of RDFS is to provide a means for defining classes, taxonomies of classes and properties, domains and ranges. All subsumption relationships of classes and properties and all domains and ranges of properties defined in RDFS are represented as the following RDFS axioms which are always true. Class or property membership relations of RDFS vocabularies are also always true and are not listed here since they are trivial.

Table 7. RDFS Vocabularies

| | |
|---|---|
| rdf:type rdfs:domain rdfs:Resource .<br>rdfs:domain rdfs:domain rdf:Property .<br>rdfs:range rdfs:domain rdf:Property .<br>rdfs:subPropertyOf rdfs:domain rdf:Property .<br>rdfs:subClassOf rdfs:domain rdfs:Class . | rdf:type rdfs:range rdfs:Class .<br>rdfs:domain rdfs:range rdfs:Class .<br>rdfs:range rdfs:range rdfs:Class .<br>rdfs:subPropertyOf rdfs:range rdf:Property .<br>rdfs:subClassOf rdfs:range rdfs:Class . |

| | |
|---|---|
| rdf:subject rdfs:domain rdf:Statement .<br>rdf:predicate rdfs:domain rdf:Statement .<br>rdf:object rdfs:domain rdf:Statement .<br>rdfs:member rdfs:domain rdfs:Resource .<br>rdf:first rdfs:domain rdf:List .<br>rdf:rest rdfs:domain rdf:List .<br>rdfs:seeAlso rdfs:domain rdfs:Resource .<br>rdfs:isDefinedBy rdfs:domain rdfs:Resource .<br>rdfs:comment rdfs:domain rdfs:Resource .<br>rdfs:label rdfs:domain rdfs:Resource .<br>rdf:value rdfs:domain rdfs:Resource .<br>rdf:_*n* rdfs:domain rdfs:Resource .<br><br>rdf:Alt rdfs:subClassOf rdfs:Container .<br>rdf:Bag rdfs:subClassOf rdfs:Container .<br>rdf:Seq rdfs:subClassOf rdfs:Container .<br>rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .<br>rdfs:Datatype rdfs:subClassOf rdfs:Class .<br>rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property . | rdf:subject rdfs:range rdfs:Resource .<br>rdf:predicate rdfs:range rdfs:Resource .<br>rdf:object rdfs:range rdfs:Resource .<br>rdfs:member rdfs:range rdfs:Resource .<br>rdf:first rdfs:range rdfs:Resource .<br>rdf:rest rdfs:range rdf:List .<br>rdfs:seeAlso rdfs:range rdfs:Resource .<br>rdfs:isDefinedBy rdfs:range rdfs:Resource .<br>rdfs:comment rdfs:range rdfs:Literal .<br>rdfs:label rdfs:range rdfs:Literal .<br>rdf:value rdfs:range rdfs:Resource .<br>rdf:_*n* rdfs:range rdfs:Resource .<br><br>rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .<br><br>rdf:XMLLiteral rdf:type rdfs:Datatype .<br>rdf:_*n* rdf:type rdfs:ContainerMembershipProperty . |

RDFS vocabularies also have a formal meaning which entails some conclusions from a given RDF graph. This meaning is represented as the following 13 RDFS entailment rules. The question marks (?) introduce variables and *isLiteral*() checks if its argument is a plain literal.

| Rule name | Conditions | Conclusions |
|---|---|---|
| rdfs1 | (?x ?a ?y) *isLiteral*(?y) | (?y rdf:type rdfs:Literal) |
| rdfs2 | (?a rdfs:domain ?x) (?y ?a ?z) | (?y rdf:type ?x) |
| rdfs3 | (?a rdfs:range ?x) (?y ?a ?z) | (?z rdf:type ?x) |
| rdfs4 | (?x ?a ?y) | (?x rdf:type rdfs:Resource)<br>(?y rdf:type rdfs:Resource) |
| rdfs5 | (?a rdfs:subPropertyOf ?b) (?b rdfs:subPropertyOf ?c) | (?a rdfs:subPropertyOf ?c) |
| rdfs6 | (?a rdf:type rdf:Property) | (?a rdfs:subPropertyOf ?a) |
| rdfs7 | (?a rdfs:subPropertyOf ?b) (?x ?a ?y) | (?x ?b ?y) |
| rdfs8 | (?x rdf:type rdfs:Class) | (?x rdfs:subClassOf rdfs:Resource) |
| rdfs9 | (?x rdfs:subClassOf ?y) (?z rdf:type ?x) | (?z rdf:type ?y) |
| rdfs10 | (?x rdf:type rdfs:Class) | (?x rdfs:subClassOf ?x) |
| rdfs11 | (?x rdfs:subClassOf ?y) (?y rdfs:subClassOf ?z) | (?x rdfs:subClassOf ?z) |
| rdfs12 | (?a rdf:type rdfs:ContainerMembershipProperty) | (?a rdfs:subPropertyOf rdfs:member) |
| rdfs13 | (?x rdf:type rdfs:Datatype) | (?x rdfs:subClassOf rdfs:Literal) |

rdfs1 represents the type of plain literal. rdfs2 and rdfs3 represent the meaning of domain and range properties (`rdfs:domain` and `rdfs:range`) respectively which restricts the types of domain and range of a property. rdfs4 indicates that subject and object of a RDF statement must be a `rdfs:Resource`. rdfs5, rdfs6 and rdfs7 say the transitive, reflexive and subsuming characters of the property-subsumption relationship, `rdfs:subPropertyOf`, respectively. rdfs8 represents that

`rdfs:Resource` is a super-class of all classes. rdfs9, rdfs10 and rdfs11 say the subsuming, reflexive and transitive characters of the class-*subsumption* relationship, `rdfs:subClassOf`, respectively. rdfs12 indicates that `rdfs:member` is a super-property of all container-membership properties. Last, rdfs13 represents that `rdfs:Literal` is a super-class of all data-type classes.

For more information about interpretation of RDF and RDFS vocabularies, please refer to the authoritative specification site [6, 17].

## *2.6. RDF/S limitations*

In RDF/S there is no notion of incorrectness or inconsistency of the knowledge represented and everything that is true at a given time will remain true later whatever new knowledge was learnt be it for the data (e.g. a new property asserted for a resource) or for the schema (e.g. a new domain added to a property). New knowledge only leads to additions. RDF/S is said to be positive, conjunctive and monotonous.

Point to OWL, RIF, POWDER
////
The previously discussed types of semantics are not the only meaningful types of semantics of RDF and RDFS. In some cases it might be of value to be able to get logical consequences that are not possible with the mentioned standard semantics (intentional semantics). In those cases an extensional semantics which has stricter requirements for interpretations can be used. Yet in general we use the standard semantics since we can easily implement its deduction rules which facilitate the development of software using RDF and RDFS. That is why this minimal set of requirements is set as standard for systems compatible to RDF and RDFS. Yet we can create systems supporting stricter types of semantics.
An example for a deduction which is meaningful but not supported by standard semantics is the following:
From

**ex:talksTo rdfs:domain ex:Homo**
**ex:Homo rdfs:subClassOf ex:Primates**

we could want to deduce

**ex:talksTo rdfs:domain ex:Primates**.

Besides that RDFS has its limitations when it comes to negations. It is not possible with RDFS to state that something does not hold. Of course one can use the names of classes or properties to express negation (**ex:nonSmoker**), however in the following example the two statements do not end in a contradiction and furthermore it is not possible to express that **ex:nonSmoker** and **ex:smoker** have no elements in common:

**ex:michael rdf:type ex:nonSmoker**
**ex:michael rdf:type ex:Smoker**

# 3. Examples and applications

## *3.1. Data integration in the wild*

The goal of Linked Data [http://linkeddata.org/] is to enable people to share structured data on the Web as easily as they can share documents today. Linked Data is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. More specifically, Wikipedia defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF."

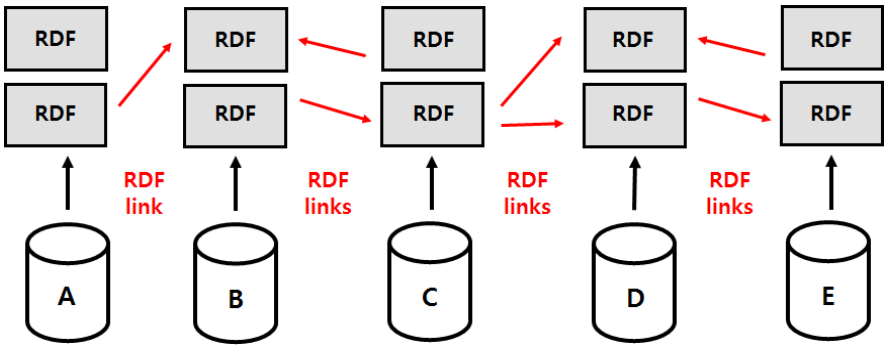



Fig. 5. Concept of Linked Data [Christian Bizer: How to Publish Linked Data on the Web – Introduction]

Linked Data is simply about using the Web to create typed links between data from different sources. The basic tenets of Linked Data [3] are to:

use the RDF data model to publish structured data on the Web

use RDF links to interlink data from different data sources

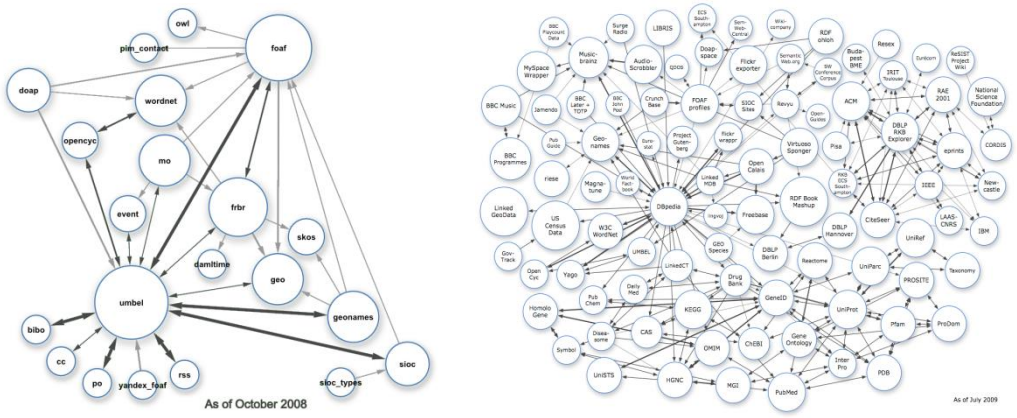



Fig. 6. Linked Data Set from October 2008 to July 2009 [http://linkeddata.org/]. The size of the circles corresponds to the number of triples in each dataset. The size of linked data set in October 2008 is over 2 billion and that of in July 2009 is over 4.5 billion.

More details on this initiative will be given in the linked data chapter.

## *3.2. Schema and data samples*

Like the web of hypertext, the web of data is constructed with documents on the web. However, unlike the web of hypertext, where links are relationships anchors in hypertext documents written in HTML, for data they link between arbitrary things described in RDF. The URIs identify any kind of object or concept. But for HTML or RDF, the same expectations apply to make the web grow:

Use RDF to publish descriptions;

Use URIs as names for things;

Use HTTP URIs so that people and software can look up those names;

When someone looks up a URI, provide useful information (use HTTP content negotiation to provide HTML pages to humans and RDF descriptions to software)

In these descriptions, include statements that link to other URIs so that related things can be discovered.

RDF/S has application to any information portal or data-intensive web site or data integration project, as its flexibility and extensibility allows opening up domain-specific data silos. To this end, RDF/S is of high interest for creating cross-domain data sets and to foster the networked economy. Prominent examples of data silos that are often made publicly available by means of ontologies and RDF data sets include, amongst many others, the following:

Documents (e.g., by means of the Dublin Core vocabulary – www.dublincore.org): annotation of title, author, creation date, or subject amongst many other properties, for instance to make digital library content usable to applications on the web, like cross-library search engines.

Schema Example:

**http://purl.org/dc/terms/Agent**

Class: A resource that acts or has the power to act.

**http://purl.org/dc/terms/contributor**

Property: An entity responsible for making contributions to the resource.

Range: http://purl.org/dc/terms/Agent

**http://purl.org/dc/terms/creator**

Property: An entity primarily responsible for making the resource.

SubPropertyOf: http://purl.org/dc/terms/contributor

Range: http://purl.org/dc/terms/Agent

**http://purl.org/dc/terms/created**

Property: Date of creation of the resource.

SubPropertyOf: http://purl.org/dc/terms/date

Range: http://www.w3.org/2000/01/rdf-schema#Literal

**http://purl.org/dc/terms/isReplacedBy**

Property: A related resource that displaces, or supersedes the described resource.

SubPropertyOf: http://purl.org/dc/terms/relation

Data Example:

There is a document Document1 with three contributors, whereof one is the main creator. The document is a DCMI Metadata example and was created on June 15 2009; subsequently it was replaced by a new document referred to as Document2.

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix ex: <http://www.example.org/> .
ex:Document1   dc:contributor ex:Contributor2, ex:Contributor3;
     dc:creator ex:Contributor1;
          dc:created "2009-06-15";
          dc:description "This is a DCMI Metadata example";
          dc:isReplacedBy ex:Document2.
```

Persons (e.g., FOAF – www.foaf-project.org; VCard – www.w3.org/Submission/vcard-rdf/): annotations of people descriptions with name, email, web page, and links to friends are used to make personal web sites intelligible to browser, and offer services to the user like saving contact details in his address book.

Schema Example:

http://xmlns.com/foaf/0.1/Person

Class: A person.

SubClassOf: http://xmlns.com/foaf/0.1/Agent

**http://xmlns.com/foaf/0.1/name**

Property: A name for some thing.

Range: http://www.w3.org/2000/01/rdf-schema#Literal

**http://xmlns.com/foaf/0.1/workplaceHomepage**

Property: The homepage of an organization a person works for.

Domain: http://xmlns.com/foaf/0.1/Person

Range: http://xmlns.com/foaf/0.1/Document

http://www.w3.org/2006/vcard/ns#VCard

Class: An electronic business card

**http://www.w3.org/2006/vcard/ns#fn**

Property: The full name of the object the vCard represents.

Domain: http://www.w3.org/2006/vcard/ns#VCard

Range: http://www.w3.org/2000/01/rdf-schema#Literal

**http://www.w3.org/2006/vcard/ns#title**

Property: Job title, functional position or function of the object the vCard represents.

Domain: http://www.w3.org/2006/vcard/ns#VCard

Range: http://www.w3.org/2000/01/rdf-schema#Literal

Data Example:

There is some person with the name Peter Example who works as System Administrator at the company whose Web site is www.example.org/myCompany.

```
@prefix vc: <http://www.w3.org/2006/vcard/ns#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>
[] a foaf:Person;
   foaf:name "Peter Example";
   foaf:workplaceHomepage <http://www.example.org/myCompany>;
   vc:title "System Administrator".
```

Organizations (FOAF; GoodRelations – www.purl.org/goodrelations/): activities, public/private, name, branches, domain for instance to automatically building and maintaining yellow pages services.

Schema Example:

**http://purl.org/goodrelations/v1#BusinessEntity**

Class: The legal agent making a particular offering.

**http://purl.org/goodrelations/v1#legalName**

Property: The legal name of the business entity.

Domain: http://purl.org/goodrelations/v1#BusinessEntity

Data Example:

There is some business entity with the legal name „The Example Company Ltd." that is located at Example Road 2 in 12345 ExampleCity. The companies email address is office@example-company.org

```
@prefix vc: <http://www.w3.org/2006/vcard/ns#>.
@prefix gr: <http://purl.org/goodrelations/v1#>
[] a gr:BusinessEntity;
   gr:legalName "The Example Company Ltd.";
   vc:email <mailto:office@example-company.org>;
   vc:adr [ vc:street "Example Road 2 ;
                    vc:locality "ExampleCity" ;
```

vc:postal-code “12345” ] .

Copyrights (e.g., Creative Commons – www.creativecommons.org/ns): license and conditions for reuse of digital works to filter search results to those that can actually be used for ones task.

Schema Example:

**http://creativecommons.org/ns#Work**

Class: A potentially copyrightable work.

**http://creativecommons.org/ns#License**

Class: A set of requests/permissions to users of a Work.

http://creativecommons.org/ns#license

Property: A Work has a license.

SubPropertyOf: http://purl.org/dc/terms/license

Domain: http://creativecommons.org/ns#Work

Range: http://creativecommons.org/ns#License

http://creativecommons.org/ns#legalcode

Property: The URL of the legal text of a License.

Domain: http://creativecommons.org/ns#License

Data Example:

There is a piece of work with a Attribution-ShareAlike 3.0 license from creative commons that allows distribution of the work under the condition that the owner is attributed. The human readable legal text of the license is published at http://creativecommons.org/licenses/by-sa/3.0/legalcode.

@prefix cc: <http://creativecommons.org/ns#>.
@prefix ex: <http://www.example.org/>.
ex:myWork cc:license
<http://creativecommons.org/licenses/by-sa/3.0/> .
<http://creativecommons.org/licenses/by-sa/3.0/>
cc:permits <http://web.resource.org/cc/Distribution>;
cc:requires <http://web.resource.org/cc/Attribution>;
cc:legalcode
<http://creativecommons.org/licenses/by-sa/3.0/legalcode> .

Products (GoodRelations; eClass – www.ebusiness-unibw.org/ontologies/eclass): prices, references, reviews, availability, shipping, etc. for instance to customize catalogs or aggregate feedbacks into benchmarks.

Schema Example:

http://purl.org/goodrelations/v1#hasUnitOfMeasurement

Property: The unit of measurement given using the UN/CEFACT Common Code

Domain: http://purl.org/goodrelations/v1#QuantitativeValue, http://purl.org/goodrelations/v1#TypeAndQuantityNode http://purl.org/goodrelations/v1#UnitPriceSpecification

Range: http://www.w3.org/2001/XMLSchema#string

http://purl.org/goodrelations/v1#hasPriceSpecification

Property: This links an offering to one or more price specifications.

Domain: http://purl.org/goodrelations/v1#Offering

Range: http://purl.org/goodrelations/v1#DeliveryChargeSpecification, http://purl.org/goodrelations/v1#PaymentChargeSpecification, http://purl.org/goodrelations/v1#PriceSpecification, http://purl.org/goodrelations/v1#UnitPriceSpecification

Data Example:

There is a product of type pencil [AKF303003] whose length [BAF559001] is 150mm.

@prefix eco:
<http://www.ebusiness-unibw.org/ontologies/eclass/5.1.4/#> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
[] a eco:C_AKF303003-gen ;
eco:P_BAF559001 [ a gr:QuantitativeValueFloat ;
gr:hasUnitOfMeasurement "MMT"^^xsd:string ;
gr:hasValueFloat "150.0"^^xsd:float .

Calendar/Events (RDF Calendar – www.w3.org/2002/12/cal/; NCAL – www.semanticdesktop.org/ontologies/ncal/): name, dates, location or durations to allow various calendar data to be integrated, imported and shared in web applications.

Schema Example:

```
http://www.w3.org/2002/12/cal/ical#Vevent
Class: A grouping of component properties that describe an event.
http://www.w3.org/2002/12/cal/ical#location
Property: Defines the intended venue for the activity defined by a calendar
component.
Domain:                          http://www.w3.org/2002/12/cal/ical#Vevent,
http://www.w3.org/2002/12/cal/ical#Vtodo
Range: http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/2002/12/cal/ical#dtstart
Property: Specifies when the calendar component begins.
Domain:                          http://www.w3.org/2002/12/cal/ical#Vevent,
http://www.w3.org/2002/12/cal/ical#Vtodo,
   http://www.w3.org/2002/12/cal/ical#Vfreebusy
Range: http://www.w3.org/2001/XMLSchema#dateTime
http://www.w3.org/2002/12/cal/ical#dtend
Property: Specifies the date and time that a calendar component ends.
Domain:                          http://www.w3.org/2002/12/cal/ical#Vevent,
http://www.w3.org/2002/12/cal/ical#Vfreebusy
Range: http://www.w3.org/2001/XMLSchema#dateTime
```

Data Example:

There is an iCal event planned for to take place at the Example Hotel in London on June 24, 2010 between 9am and 6pm.

@prefix cal: < http://www.w3.org/2002/12/cal/ical#> .
[] a cal:Vevent;
cal:location “Example Hotel, London, UK”;
cal:dtstart “2010-06-24T09:00:00”;
cal:dtend “2010-06-24T18:00:00” .

Places (GeoNames – www.geonames.org/ontology/; WGS84 – www.w3.org/2003/01/geo/wgs84_pos): geographical locations, coordinates, countries to combine geo data with mapping views or use location and places as filtering criterion in web searches.

Schema Example:

```
http://www.geonames.org/ontology#Feature
Class: A geographical object uniquely defined by its geonames id.
http://www.geonames.org/ontology#name
Property: The preferred name of the feature.
Domain: http://www.geonames.org/ontology#Feature
http://www.geonames.org/ontology#countryCode
```

```
Property: A two letters country code in the ISO 3166 list
Domain: http://www.geonames.org/ontology#Feature
http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
Class: Anything with spatial extent
http://www.w3.org/2003/01/geo/wgs84_pos#lat
Property: The WGS84 latitude of a SpatialThing
Domain: http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
http://www.w3.org/2003/01/geo/wgs84_pos#long
Property: The WGS84 longitude of a SpatialThing
Domain: http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing
```

Data Example:

The resource with geonames ID 2643743 represents London in the UK, given by the ISO country code GB. The geographic location is given by the WSG84 latitude/longitude coordinates 51.5005/-0.1288.

```
@prefix geo:        <http://www.geonames.org/ontology#> .
@prefix wsg: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
<http://sws.geonames.org/2643743/>        geo:name “London” ;
                                          geo:countryCode “GB” ;
                                          wsg:lat “51.5005149421307” ;
                                          wsg:long “-0.12883186340332” .
```

Social networks (SIOC – www.sioc-project.org; RELATIONSHIP– vocab.org/relationship/): to go beyond the silo architecture and allow connections and interoperability across web 2.0 platforms.

Schema Example:

```
http://rdfs.org/sioc/ns#Community
   Class: A high-level concept that defines an online community and what it
consists of.
```

**http://rdfs.org/sioc/ns#Post**

```
   Class: An article or message that can be posted to a Forum.
   SubClassOf: http://rdfs.org/sioc/ns#Item
http://rdfs.org/sioc/ns#has_creator
   Property: This is the User who made this Item.
   Domain: http://rdfs.org/sioc/ns#Item
   Range: http://rdfs.org/sioc/ns#User
http://rdfs.org/sioc/ns#topic
   Property: A topic of interest; e.g., in the Open Directory Project or of
a SKOS category.
   SubPropertyOf: http://purl.org/dc/terms/subject
```

**http://rdfs.org/sioc/ns#has_reply**

```
   Property: Points to an Item that is a reply or response to this Item.
   Domain: http://rdfs.org/sioc/ns#Item
   Range: http://rdfs.org/sioc/ns#Item
```

Data Example:

There is a post from September 7, 2006 by the author identified by <example.org/author> on the topic of annotation. There is a comment in reply to this post.

```
@prefix sioc:        <http://rdfs.org/sioc/ns#> .
@prefix dc: <http://purl.org/dc/terms/> .
<http://example.org/blog/2010/entry> a sioc:Post ;
   dc:created “2006-09-07T09:33:30Z” ;
   sioc:has_creator <http://example.com/author/> ;
   sioc:topic <http://example.org/topics/annotation> ;
```

```
sioc:has_reply <http://example.org/blog/2010/entry#comment-1> .
```

Lexicons, Index, Folksonomies (SKOS – www.w3.org/2004/02/skos/, NiceTag - http://ns.inria.fr/nicetag/2010/07/21/voc): semantic descriptions of thesauri, classification schemes, subject heading lists and taxonomies.

Schema Example:

```
http://www.w3.org/2004/02/skos/core#Concept
   Class: A concept in a knowledge organization system
http://www.w3.org/2004/02/skos/core#OrderedCollection
   Class: An ordered collection of concepts
http://www.w3.org/2004/02/skos/core#prefLabel
   Property: The preferred lexical label for a resource, in a given
language.
   SubPropertyOf: http://www.w3.org/2000/01/rdf-schema#label
http://www.w3.org/2004/02/skos/core#narrower
   Property: Relates a concept to a concept that is more specific in
meaning.
http://www.w3.org/2004/02/skos/core#broader
   Property: Relates a concept to a concept that is more general in
meaning.
http://www.w3.org/2004/02/skos/core#related
   Property: Relates concepts for which there is an associative semantic
relationship.
http://ns.inria.fr/nicetag/2010/07/21/voc#AnnotatedResource
   Class: A dereferenceable resource on the Web that is taggable
http://ns.inria.fr/nicetag/2010/07/21/voc#isRelevantToSt
   Property: Links a resource to anything that it may be relevant to.
```

Data Example:

The topic annotation is related to the concepts metadata and footnote, where footnote is a narrower term than annotation. The annotation concept is moreover described as being relevant to the book chapter on RDF/S.

```
@prefix skos:      <http://www.w3.org/2004/02/skos/core#> .
@prefix voc:       <http://ns.inria.fr/nicetag/2010/07/21/voc#> .
<http://example.org/topics/annotation> a skos:Concept;
    skos:definition “Any descriptive notation applied to data”;
    skos:prefLabel “Annotation”;
    skos:narrower <http://example.org/topics/footnote>;
    skos:related <http://example.org/topics/metadata> ;
    a voc:AnnotatedResource ;
    voc:isRelevantToSt <http://example.org/bookchapter/RDFS> .
```

Genetics and Life Science (e.g. Gene Ontology – www.geneontology.org; Open Biomedical Ontology – www.obofoundry.org; OMIM – www.ncbi.nlm.nih.gov/omim): molecular functions, biological processes, cellular components and vocabularies to describe the human anatomy and genes, biochemistry, genetic disorders or phenotypes.

Schema Example:

```
http://www.geneontology.org/dtd/go.dtd#term
   Class: Any term in the gene ontology is of type term
http://www.geneontology.org/dtd/go.dtd#is_a
   Property: If A is_a B, then A is a subtype of B.
   Domain: http://www.geneontology.org/dtd/go.dtd#term
   Range: http://www.geneontology.org/dtd/go.dtd#term
```

**http://www.geneontology.org/dtd/go.dtd#part_of**

`Property: Representation of part-whole relationships.`

**http://www.geneontology.org/dtd/go.dtd# regulates**

`Property: One process directly affects the manifestation of another process or quality.`

Data Example:

The term GO:0000001 represents the concept of mitochondrion inheritance which is a subtype of the concept GO:0048308 (organelle inheritance), a part of the concept GO:0009530 (primary cell wall), and negatively regulate GO:0006312 (mitotic recombination).

@prefix godtd: <http://www.geneontology.org/dtd/go.dtd#> .
@prefix go: <http://www.geneontology.org/go#> .
go:GO:0000001 a go:term ;
godtd:accession "GO:0000001" ;
godtd:name "mitochondrion inheritance" ;
godtd:synonym "mitochondrial inheritance" ;
godtd:definition
"The distribution of mitochondria, including the mitochondrial genome, into daughter cells after mitosis or meiosis, mediated by interactions between mitochondria and the cytoskeleton." ;
godtd:is_a go:GO:0048308 ;
godtd:part_of go:GO:0009530 ;
godtd:negatively_regulates go:GO:0006312 .

Healthcare (SNOMED CT – www.ihtsdo.org/snomed-ct/; MeSH – www.nlm.nih.gov/mesh; UMLS – www.nlm.nih.gov/research/umls/): clinical and disease-related aspects, medications, treatments and health records for the integration of patient records and the optimization of medical information flows.

## *3.3. RDF Web application examples*

Although the listing above presents mostly independent vocabularies and data sets, the strength of RDF lies in the flexibility of integration. RDF provides, in contrast to relational data models, a schema-free data model. RDF graphs can in principle quite easily be merged by sharing particular resources, or claiming two resources to be the same, although their identifier might be different. The integration approach that RDF offers is driven by the linking of resources in subjects and objects of triples. In fact, linking resources by means of URIs is one of the four principles of the Linked Open Data (LOD) initiative. Reconsidering the listing above, RDF allows, for example, to link the description of a document to the description of its author (person), linking the author to the description of her affiliation (organization) and colleagues, linking the organization to the description of the geographical location that is linked to transportation plans, hotels, events and the history and demographics of the place; from the history there would be links to important personalities which are linked to events, groups and biographical data, and so on.

RDF not only provides this ability to publish and link the data, it also provides the foundational shared data model on which other capabilities are built: querying this data (SPARQL is built on top of RDF), embedding (RDFa and GRDDL rely on the RDF model), and inference and reasoning (RDFS and OWL are defined on top of RDF, even RIF has a part dealing with RDF). Before delving into such details about RDF and related technologies and languages, this chapter concludes with a review of some of the most well-known applications that are built on top of RDF/S.

The BBC web site (Figure 7) creates web identifiers for every item to enable very rich cross-domain aggregation. The content can be discovered by users in many different ways. The RDF representations allow developers to use BBC data to build richer applications, including new BBC products since the web site becomes an API and their development becomes loosely coupled. As an example a web identifier is provided for every species, habitat and adaptation mentioned in the BBC

programs. Data is aggregated from Wikipedia, the Animal Diversity Web, WWF's Wildfinder, the Zoological Society of London's EDGE of Existence program, and the IUCN's Red List of Threatened Species. This data is then reintegrated and linked out to multimedia resources from the BBC.


Fig. 7. Linking data from external sources to program clips extracted from the BBC's archives [25]

In May 2008 Yahoo! launched SearchMonkey (Figure 8) an open developer platform for search that allows developers to build applications on top of their search engine and in particular allowing site owners to share structured data (RDF/S and RDFa) with the engine and customize the search results presentations. For instance, a result found on LinkedIn will include name, surname, position of the person, a result found on YouTube will include the title and a direct access to the video, a result found on Amazon will include the Title, the author and the average review, etc.

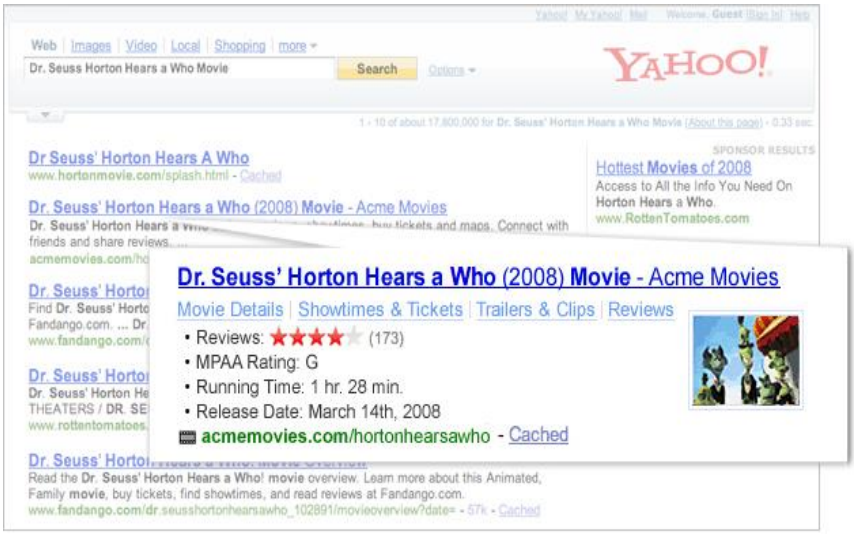



Fig. 8. SearchMonkey developer tool to extract data and build apps to display custom enhanced results by Yahoo! [29]

Open Calais from Thomson Reuters is a semantic web service and API (Figure 9). For any document submitted into Open Calais, entities are identified, extracted and annotated in RDF. Using these annotations a large number of functionalities can be supported and improved: more precise searching, subject monitoring, custom alerts and notification, thematic routing of information,

intelligence, link suggestion and augmented browsing on the fly. More on this type of approaches will be said in the automatic annotation chapter.

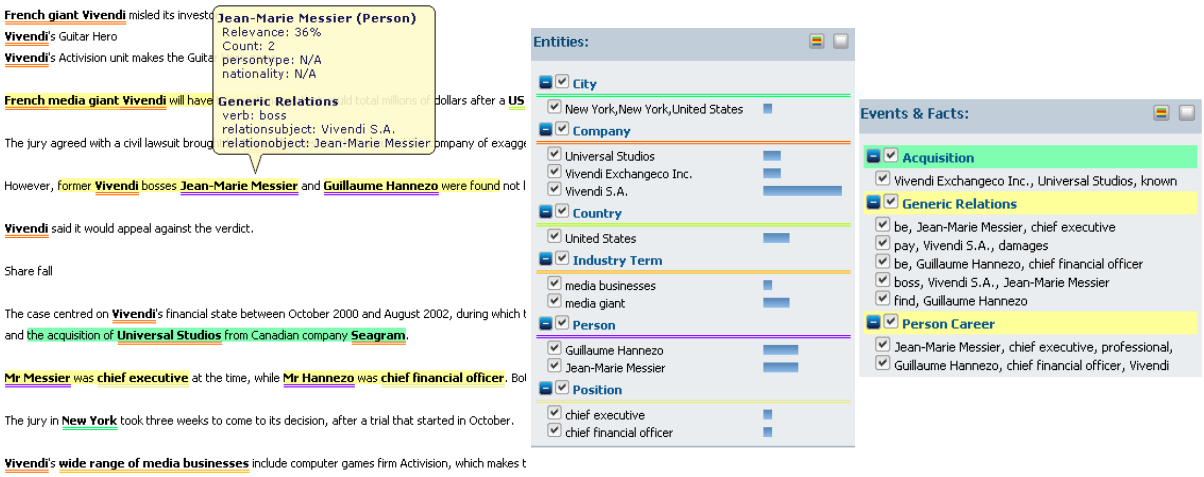



Fig. 9. The OpenCalais Web Service automatically creates rich semantic metadata from text[30]

Zemanta (Figure 10) is an API relying on RDF and used in a blogging tool which suggests related content and pictures to the posts and automatically links it to other online resources. The same API is also used by other services including the semantic bookmarking service Faviki.

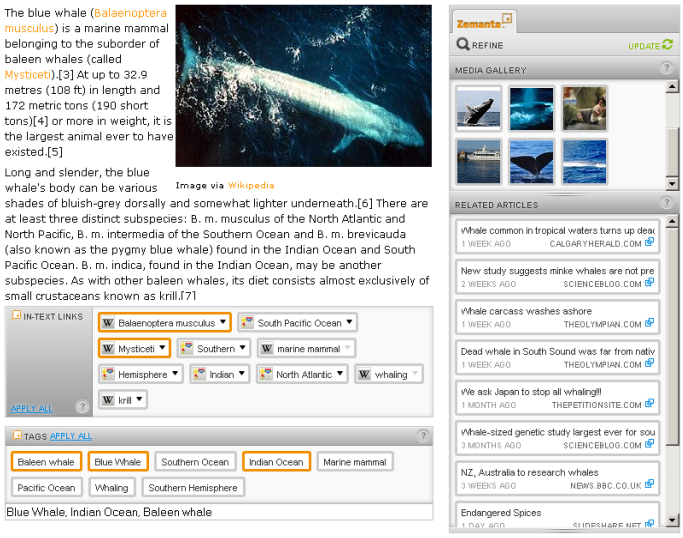



Fig. 10 Any text posted in Zemanta [31] it gets augmented and linked to other resources automatically.

Creative Commons (Figure 11) is a nonprofit corporation dedicated to making it easier to share and build upon the work found online while respecting their copyright. They provide free licenses to mark (in RDF) creative work with the freedom the creator wants it to carry, so others can share, remix, use commercially, or any combination thereof.

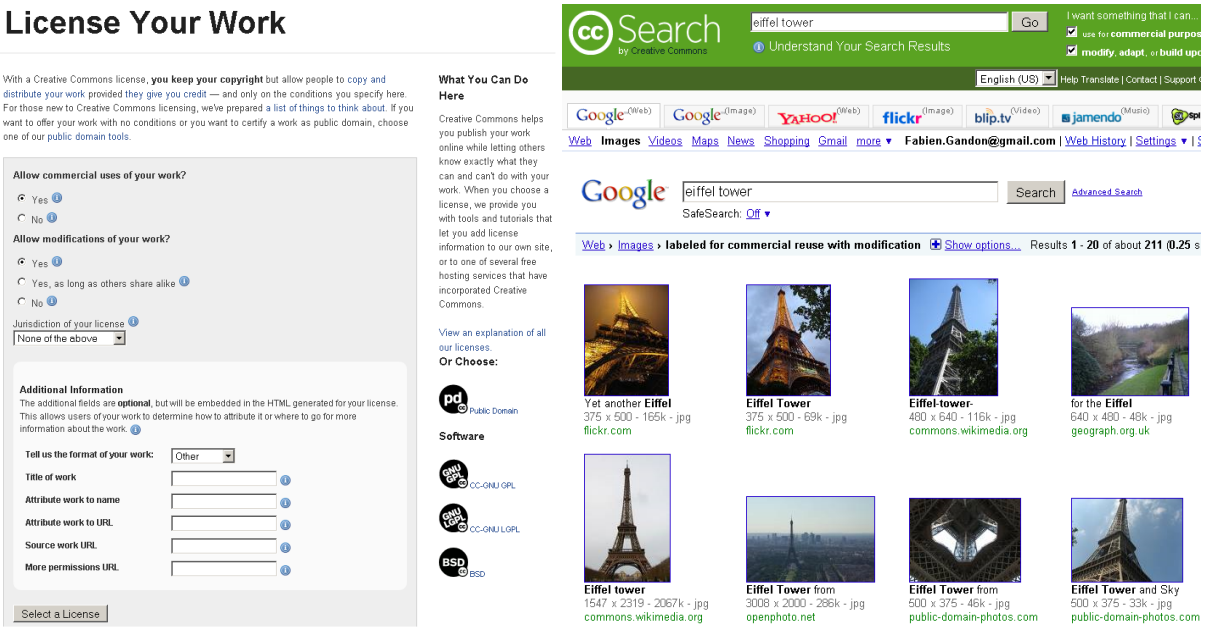



Fig. 11. Licensing creative work and searching for reusable resources with Creative Commons[26]

Cambridge Semantics (Figure 12) is a provider of semantic middleware and application development tools. They focus on providing a single, universal layer for representing, accessing and combining enterprise data on-the-fly. Anzo suite is a web-based collaboration framework for browsing and sharing data in real time. At the heart of the Anzo suite a server acts as a central gateway that provides a consistent interface for applications to read, write, and query RDF data, regardless of the actual source of the data.

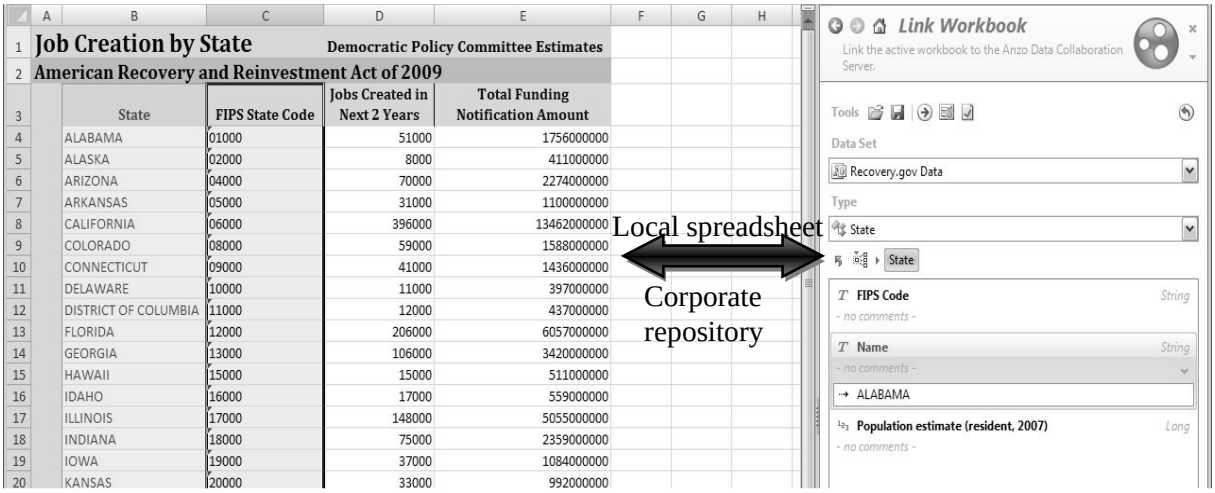



Fig. 12. A Microsoft Excel plug-in by Cambridge Semantics that allows spreadsheet data to be shared, discovered, and reused in other spreadsheets, on the Web, and in relational databases

OntoFrame (Figure 13), a semantic service platform developed by KISTI [27] since 2005, aims to provide an infrastructure for evolutionary search services based on semantic knowledge management, information retrieval, and reasoning technologies. A semantic portal service of academic research information [http://www.w3.org/2001/sw/sweo/public/UseCases/OntoFrame/] using OntoFrame is now available as a beta service on the Web. The reasoning engine named OntoReasoner in the platform fully covers RDF/S with a state-of-the-art query performance. KISTI has applied their technologies to practical domains such as intelligent legislation support system of Ministry of Justice

Republic of Korea, standard information system of Korea Agency for Technology and Standards, and the experts recommendation system of National Research Foundation of Korea.

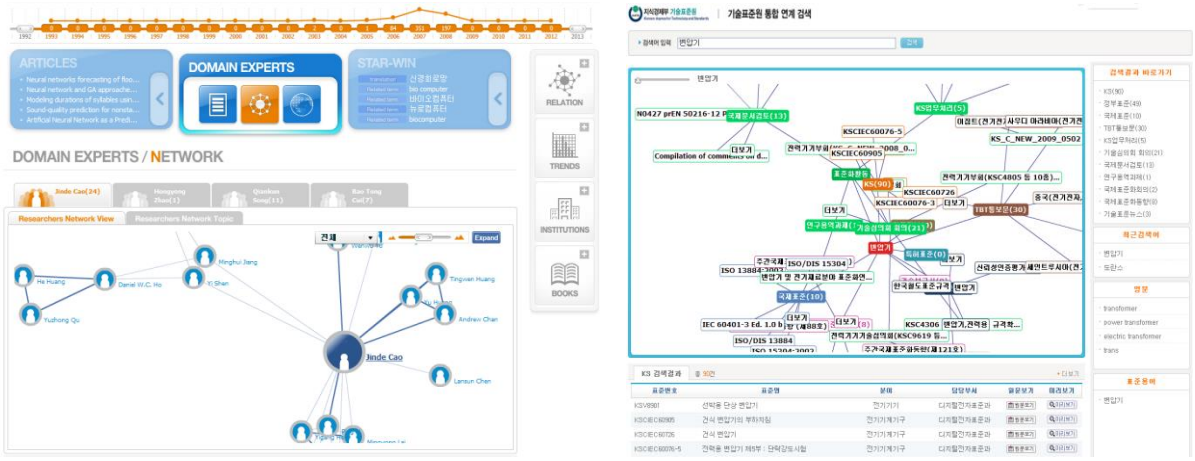



Fig. 13. OntoFrame-based services with search and reasoning functions (left: researcher network service in OntoFrame S3, right: standard information browsing in OntoFrame-KATS)

A number of other applications are also available on the Web. DBpedia [http://dbpedia.org/], an effort to extract structured information from Wikipedia and make it available on the Web in RDF. It allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data.

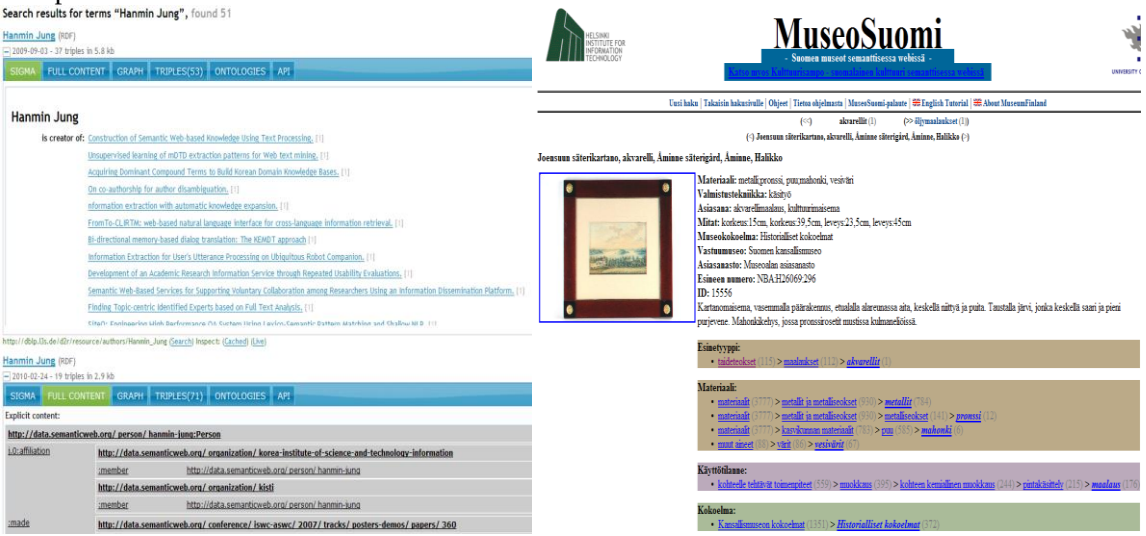



Fig. 14. (a) An example of semantic data search results with Sindice, which shows a view of what seem to be the "main topic", full content, RDF triples, ontologies used by URIs and so on. (b) Search results for a material, including metadata and faceted categories such as user, place of use, situation of use, and museum collection in MuseoSuomi

Sindice [http://sindice.com/] as a semantic web index and search engine (Figure 14) embeds hundreds of millions web pages with RDF and Microformats to find billion pieces of reusable information. MuseoSuomi [http://www.museosuomi.fi/], an application publishing data about cultural collections (Figure 14), is presented from the viewpoints of the end-user and the museums providing the contents. Through semantic web techniques, it is possible to make collections semantically interoperable and provide museum visitors with intelligent content-based search and browsing services to the global collection base. KmP (Figure 15) is a real world experiment on the design and usages of a

customizable semantic web server to generate up-to-date views of the Telecom Valley and assist the management of competencies at the level of the organizations (companies, research institute and labs, clubs, associations, government agencies, schools and universities, etc.). The KmP platform aims at increasing the portfolio of competences of the technological pole of Sophia Antipolis by implementing a public knowledge management solution at the scale of the Telecom Valley based on a shared repository and a common language to describe and compare the needs and the resources of all the organizations. It resulted in a portal that relies on a semantic web server publicly available for all the actors of the value chain of the Telecom Valley of Sophia Antipolis.

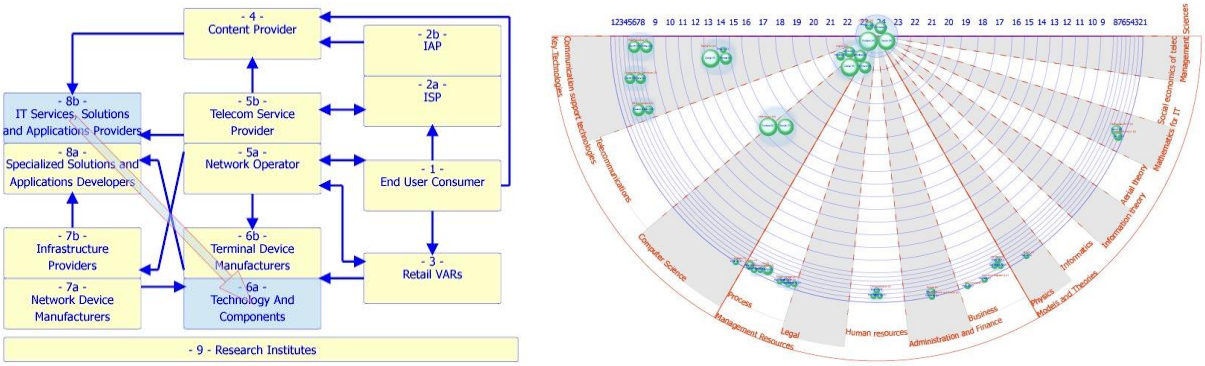



Fig. 15. SVG rendering of exchanges on the value chain of the Telecom Valley of Sophia Antipolis and radar view of the clusters of competencies.

DERI Pipes [http://pipes.deri.org/] is an open source extensible mashups platform (Figure 16) supporting RDF, XML, Microformats, JSON, and binary streams. As a 'Web Pipe', it produces output streams of data that can be used by applications. OntoPipeliner [28], as another semantic mashup tool, dynamically assembles existing semantically-operated services to help the user to design a new service.

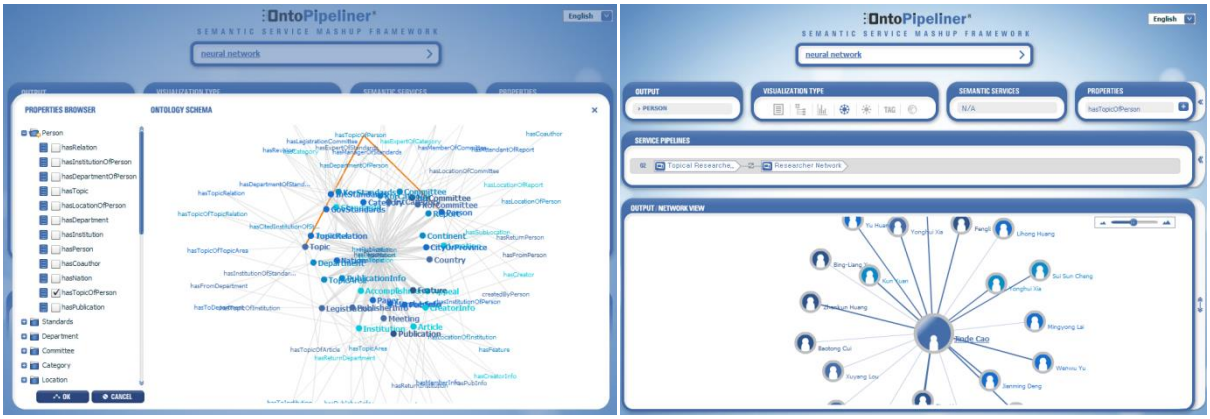



Fig. 16. Mash up of semantically-operated services under several constraints such as properties, visualization type, and output class (left: selecting properties for constraint setting, right: mashup results generated from two services)

**Related Resources**

Books

*A Semantic Web Primer* (2nd Edition) by Grigoris Antoniou and Frank van Harmelen (ISBN-13: 978-0262012423, MIT Press): The primer provides a systematic approach to the different languages (e.g., XML, RDF, and OWL) and technologies (ontologies, logics and inference) that are central to Semantic Web development. The second edition includes amongst many other extension new material

on SPARQL. Supplementary materials, including slides, online versions of many of the code fragments in the book, and links to further reading, can be found at www.semanticwebprimer.org.

*Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* by Dean Allemang and James Hendler (ISBN-13: 978-0123735560, Morgan Kaufmann): This book offers insightful information to anyone who works with managing, sharing and accessing information. It provides a solid base of knowledge about the principles and technologies, as well as main architectural building blocks of the Semantic Web. Additionally it addresses more advanced topics such as inference, ontology languages and various Do's and Don'ts of ontology engineering. In summary, the book is considered of interest to readers that wonder about the technicalities leveraging the future of data management on the Web.

*Foundations of Semantic Web Technologies* by Pascal Hitzler, Markus Krötzsch and Sebastian Rudolph (ISBN-13: 978-1420090505, Chapman & Hall): This book focuses on ontology languages that are standardized or under standardization mostly by the W3C: RDF Schema, OWL, Rules, and the corresponding query languages such as SPARQL. Interesting aspects of the book are the chapters about the very recent developments around
OWL 2 and the Rule Interchange Format (RIF).

*Semantic Web Technologies: Trends and Research in Ontology-based Systems* by John Davies, Rudi Studer and Paul Warren (ISBN-13: 978-0470025963, Wiley): First of all, this book provides a extensive introduction to (semi-)automatic ontology generation and metadata extraction. In this context, the authors report about ontology management and evolution, reasoning with inconsistent ontologies, ontology mediation and semantic information access. The final chapters examine how Semantic Web technology is being applied in digital libraries, the telecom industry and in Semantic Web services.

*The Semantic Web. Real-world Applications from Industry* by Jorge Cardoso, Martin Hepp and Miltiadis Lytras (ISBN-13: 978-0387485300, Springer): This book provides a series of case studies which give examples of how real benefits can be achieved by adopting semantic technologies in real world situations. Examples are collected from finance and government, B2B integration, tourism, education, healthcare and others. This mixture of expert contributions provides not only valuable technical details but a wealth of lessons learned.

Software and Tools

Protégé (protege.standford.edu): Protégé is a free, open source Java ontology editor and knowledge base framework. Its success is largely based on the extensible plug-and-play environment that makes it a flexible base for rapid prototyping and application development. Protégé supports RDF/S with appropriate import and export functionalities.

NeOn Toolkit (www.neon-toolkit.org): The NeOn Toolkit is a multi-platform ontology engineering environment. The toolkit is based on the Eclipse platform, and provides plug-ins covering a variety of ontology engineering activities, such as ontology evaluation and matching, or reasoning and inference. The RDF Editor is a modeling tool for the creation and maintenance of semantic models in RDF.

RDF2Go (rdf2go.semweb4j.org): RDF2Go is an abstraction over triple (and quad) stores that allows developers of semantic applications to program against rdf2go interfaces and choose or change the underlying repository implementation later easily. There are adapters available, amongst others, for Sesame, Jena and Owlim.

Sesame (www.openrdf.org): Sesame is an open source RDF framework in Java for storage, RDFS inferencing and querying via SeRQL and SPARQL. As part of Sesame, the RIO package offers a number of RDF parsers and serializers.

Jena (jena.sourceforge.net): Jena is a Semantic Web framework for Java. It provides a programmatic environment for RDF, RDFS, OWL, and a query engine for SPARQL (www.openjena.org/ARQ/); moreover, it includes a rule-based inference engine.

Owlim (www.ontotext.com/owlim/): Owlim provides a family of semantic repositories; i.e., RDF database management systems. The Owlim RDF engine is implemented in Java and fully compliant with Sesame. It has inference support for RDFS, OWL Horst and OWL 2 RL. According to the claims of its developers, Owlim is currently the most scalability RDF repository with the best performance figure in terms of loading and query evaluation.

Virtuoso (virtuoso.openlinksw.com): Virtuoso is an enterprise grade multi-model data server with support for various data formats, amongst many others RDF too. Virtuoso includes an RDF store, a SPARQL query engine with SQL integration and various user front-ends to access and manage semantic data. There is an open source edition of Virtuoso often referred to as OpenLink Virtuoso (http://sourceforge.net/projects/virtuoso/).

AllegroGraph (http://www.franz.com/agraph/allegrograph/): The AllegroGraph RDF store offers a persistent RDF graph database with Enterprise Online Transaction Processing. AllegroGraph 4.0 supports SPARQL, RDFS++, and Prolog reasoning, and enhances this setting with temporal reasoning rules and a geospatial engine for complex event processing. There is a free server edition available that is limited to up to 50 million triples only.

Corese/KGRAM (http://www-sop.inria.fr/edelweiss/software/corese/) is a RDF engine based on Conceptual Graphs (CG). It enables the processing of RDFS, OWL Lite and RDF statements and it can perform SPARQL Queries and run rules over the RDF graph. It relies on CG formalism to deeply exploit the graph nature of RDF and investigate graph-based extensions to the formalisms and the reasoning. This engine is developed in Java and focuses on providing an efficient standalone main-memory platform used in many research and development programs.

Further tools relevant to RDF are listed at W3C's RDF Web site: http://www.w3.org/RDF/

Web sites

http://www.w3.org/RDF/: This is the main page of W3C for all matters related to the Resource Description Framework. It refers to all relevant readings and lists various RDF tools and related standards.

www.w3.org/2001/sw: This is the W3C Semantic Web Activity Web site and as such the main access point to all work done at W3C in regards to semantic technologies.

www.w3.org/TR/rdf-primer/: The RDF Primer by Frank Manola and Eric Miller is the starting point for getting to know more about RDF. It introduces its basic concepts and its XML syntax. It also explains how to define RDFS vocabularies and describes the content and purpose of the RDF specification documents.

www.linkeddata.org: The Linked Open Data community Web site provides a home for resources and information to promote and foster the Linked Data movement.

Standards

RDF/S : http://www.w3.org/standards/techs/rdf
Concepts and Abstract Syntax: http://www.w3.org/TR/rdf-concepts/
Semantics: http://www.w3.org/TR/rdf-mt/
Primer: http://www.w3.org/TR/rdf-primer/
RDF Schema: http://www.w3.org/TR/rdf-schema/
RDF/XML Syntax: http://www.w3.org/TR/rdf-syntax-grammar/

## 4. Future Issues

At the time of writing this chapter several work items have been recently identified for what could be the next version of RDF. Two major evolutions respectively concern the model of RDF and the syntax of RDF:

Graph identification and metadata on RDF graphs are widely done extensions with numerous use cases including: provenance, authorship, creation date and use-by date, accuracy, authentication, certification, validity, access rights, copyrights and many other informational context properties. This extension of RDF 1.0 requires both an evolution of the RDF model to include the notion of graph and a mechanism to name them, and an evolution of the syntax to allow the naming.

As we saw, Turtle is a widely used syntax for RDF and therefore there is a strong demand to standardize it to ensure compatibility across implementations. In addition, several JSON formats and implementations already exist and call for a standard specification of the way to serialize RDF graphs in JSON. This would provide web developers with an easier way to use RDF data within existing web tools.

One of the challenges for RDF and RDFS is now to have saleable, supported and widely deployed infrastructures (triple stores) to easily integrate industrial solutions and reach its full potential.

RDF2RDB

## 5. Summary

Semantic web is a web to link data and share the semantics of their schemas. RDF provides the first recommendation to publish and link data. RDFS provides the first recommendation to share the semantics of their schemas. RDF reuses the web approach to identify resources (URI) and to allow to explicitly represent any relationship between two resources. Such statements can come from any source on the web and be merged with other statements supporting world-wide data integration. Using and reusing URIs, anyone can say anything about any topic, anyone can add to it, and so on. RDF Schema is a typing system to describe classes of RDF resources, their properties and the relations between them. Classes and properties are viewed as sets and RDFS relies on set inclusion, intersection and union to define the semantics of these relations. RDF Schemas are written in RDF and inherit from all its advantages: distributed, extensible, same query language, etc.

The couple RDF/S (RDF & RDFS) lays the ground for all the other activities of the semantic web and is also reused in other activities of the W3C.

**References**

[1] Baader F., McGuinness D., Nardi D., and Patel-Schneider P., *The Description Logic Handbook: Theory, implementation and applications*, Cambridge University Press, Cambridge, United Kingdom, 2003.
[2] Broekstra J., Klein M., Decker S., Fensel D., van Harmelen F., and Horrocks I., *Enabling Knowledge Representation on the Web by Extending RDF Schema*, In: Shen VY, Saito N. (eds), 10th International World Wide Web Conference (WWW10), Hong Kong, pp 467–478, 2001.
[3] Bizer Ch., How to Publish Linked Data on the Web, http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/
[4] Conen W., and Klapsing R., *A Logical Interpretation of RDF* (discussion paper), Linköping Electronic Articles in Computer and Information Science, Journal of Electronic Transactions on Artificial Intelligence, 5(13), 2000.
[5] Beckett D., *Resource Description Framework (RDF) Resource Guide*, http://planetrdf.com/guide/
[6] Beckett D. (Editor), *RDF/XML Syntax Specification (Revised)*, W3C Recommendation, February 2004 at http://www.w3.org/TR/rdf-syntax-grammar/.
[7] Beckett D., and Berners-Lee T., *Turtle – Terse RDF Triple Language*, W3C Team Submission, January 2008 at http://www.w3.org/TeamSubmission/turtle/.
[8] Fensel D., Hendler J., Lieberman H., and Wahlster W. (Editors), *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, Cambridge, MA: MIT Press, 2003.
[9] Hyvönen E., Junnila M., Kettula S., Mäkelä E., Saarela S., Salminen M., Syreeni A., Valo A., and Viljanen K., *Finnish Museums on the Semantic Web: The user's Perspective on MuseumFinland*, in Proceedings of Museums and the Web Conference, 2004.
[10] Gandon F., Corby O., Giboin A., Gronnier N., and Guigard C., *Graph-based inferences in a Semantic Web Server for the Cartography of Competencies in a Telecom Valley*, ISWC, Lecture Notes in Computer Science, Galway, 2005.
[11] Gómez-Pérez A., and Corcho O., *Ontology Languages for the Semantic Web*, IEEE Intelligent Systems & their applications, 17(1):54–60, 2002.
[12] Jung H., Lee M.-K., and Sung W.-K., *Toward Semantic Web-based Law Information Management System*, Proceedings of International e-Justice and e-Law Conference, 2008.
[13] Jung H., Lee S., Kim P., Lee M., and You B.-J., *OntoPipeliner: A Semantic Broker-based Manager for Pipelining Semantically-operated Services*, Proceedings of the 8th International Semantic Web Conference (ISWC 2009), 2009.
[14] Harth, A., and Decker, S., *Optimized Index Structures for Querying RDF from the Web*, Proceedings of the 3rd Latin American Web Congress, 2005.
[15] Cardoso J., Hepp M., and Lytras M.D. (Editors): The Semantic Web: Real-World Applications from Industry. Semantic Web And Beyond Computing for Human Experience, Springer, 2007.
[16] Manola F., and Miller E., *RDF Primer: W3C Working Draft*, 2003; http://www.w3.org/TR/rdf-primer/
[17] Hayes P. (Editor), RDF Semantics, W3C Recommendation, http://www.w3.org/TR/rdf-mt/.
[18] Lee S., Jung H., Kim P., Lee M.-K., and You B.-J., *Dynamically Materializing Wild Pattern Rules Referring to Ontology Schema in Rete Framework*, Proceedings of the 1st Asian Workshop on Scalable Semantic Data Processing (AS2DP) in ASWC2009, 2009.
[19] Sintek M., and Decker S., *TRIPLE - An RDF Query, Inference, and Transformation Language*, In: Seipel D. (ed) Deductive Databases and Knowledge Management (DDLP'01), Tokyo, Japan, pp 364–378, 2001
[20] Berners-Lee T. (Editor), *Notation 3*, at http://www.w3.org/DesignIssues/Notation3.html.
[21] N-Triples, http://www.w3.org/TR/rdf-testcases/#ntriples.
[22] W3C Web standards, http://www.w3.org/2004/10/RecsFigure.png
[23] W3C, Cool URIs don't change, http://www.w3.org/Provider/Style/URI
[24] RDF lecture, http://www-asm.nii.ac.jp/~koide/SWCLOS2/Manual/RDFSchema.html
[25] W3C Use Cases, http://www.w3.org/2001/sw/sweo/public/UseCases/BBC/
[26] Creative Commons, http://creativecommons.org/
[27] KISTI, Korea Institute of Science and Technology Information, http://www.kisti.re.kr/english/
[28] OntoFrame, http://ontoframe.kr/OntoPipeliner/main.html#
[29] Yahoo SearchMonkey, http://developer.yahoo.com/searchmonkey/
[30] OpenCalais, http://www.opencalais.com/about

[31] Zemanta, http://www.zemanta.com
[32] W3C Semantic Web Best Practices and Deployment Working Group, http://www.w3.org/2001/sw/BestPractices/

Won-Kyung Sung, Hanmin Jung, Pyung Kim, In-Su Kang, Seungwoo Lee, Mi-Kyoung Lee, Dong-In Park, and Sun-Hwa Hahn, *A Semantic Portal for Researchers Using OntoFrame*, Proceedings of the 6th International Semantic Web Conference, 2007.

**Keywords:**
**Resource Description Framework (RDF), RDF Schema, Semantic Web, Ontology, metadata, Linked Open Data (LOD), semantic annotation, inference**